Quantification and Traceability of Requirements

Petter L. H. Eide

TDT4735 Software Engineering Depth Study
Fall 2005

Teaching supervisor:
Tor Stålhane

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology, Mathematics
and Electrical Engineering
Department of Computer and Information Science

Abstract

Today, software has become a critical competitive factor for many organizations. The ability to produce software of high quality, for less cost, in shorter timescales is a matter of survival. The use of requirement management tools, with support for quantification and traceability, is recognized as a significant capability in the software development and maintenance process, and as an important factor for the quality of the final software product.

This project explores the domain of requirements engineering from a technical perspective, resulting in a design and implementation of a tool for managing requirements, the ReMaTo tool.

The report presents in a brief manner theoretical concepts central to the technological aspects of requirements engineering. *Quantification* in this context is the ability to identify measurable, mutually properties of similar requirements and the elicitation of corresponding values for each requirement. *Traceability* in this context is the ability to relate artifacts which are created during the development of a software system with each other, the stakeholders that created them, and/or the rationale underpinning their exact form.

The report is intended for practitioners, managers, researchers and developers with interests in requirement management technology and tools.

PREFACE

This project has been carried out by Petter L. H. Eide from August to December 2005. The studies performed are related to the course TDT4735 Software Engineering Depth Study, being a part of the fifth year of the Master of Technology degree in Computer Sciences at the Norwegian University of Science and Technology.

The project is assigned by the WesternGeco and has been carried out at the Department of Computer and Information Science and WesternGeco Oslo Technology Center. The project is a part of the ModelWare IST research project, which is co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). Related to the ModelWare project, a prototype of the tool developed in this project was demonstrated at the "European Conference on Model Driven Architecture - Foundations and Applications", November 7-10th 2005, Nuremberg, Germany.

ACKNOWLEDGMENTS

The author would like to thank the supervisors Tor Stålhane form IDI and Bjørn Nordmoen from WesternGeco for their guidance and sharing of expertise during the writing of this report. Their feedback and council during this fall have been invaluable.

The author would also like to thank Ole-André Ranvik at Objectware and Tor Neple at Sintef for support, valuable advices and interesting viewpoints.

Trondheim, December 22, 2005

Petter L. H. Eide

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

INTRODUCTION

This chapter presents the background and foundation for the project. It starts out with a section about motivation before it focuses on the problem definition, the project context and the readers guide to the next chapters.

BACKGROUND

In the wake of the establishment of software engineering (SE) as term for the software development process, requirements engineering (RE) has gradually been formed as a research area of its own over the recent years. For the early systems, the focus of requirements was centered around the technical issues of machines being expensive resources [19]. Today, this is often characterized as constraints. Later on the attention changed towards code, design, and further on to issues connected to the elicitation of such requirements.

Issues related to eliciting requirements are still important today, but it has become more accepted that eliciting the perfect requirements is difficult and sometimes impossible at an early stage of the software development process. As a result of this, requirement change management has become a central part of the focus of RE [31]. Connected to this is validation of requirements. A common strategy to meet these challenges is using tools with support for traceability among software artifacts. This also applies to long living applications (many years), which with a large certainty will be exposed to changes.

Further, the focus of requirements engineering has changed towards "quality", ensuring that the requirements are realized in design and code, and visa versa. An extension of validation of requirements involves testing; all requirements should be addressed in one or several tests. Another challenge is the interpretation of software requirements. Using quantifying templates is a strategy to address this.

To which extent software developing companies or teams are aware of the many challenges and how they try to cope with them, aware or not, is fluctuating. This also includes the extent of use of supporting tools.

Research on quantification has been mainly concerned with categorization of requirements to be able to define common attributes for similar requirements. Further on how to make these attributes measurable and unambiguous. There exist today several well-known taxonomies for requirements.

Research on software traceability on the other hand, has been mainly concerned with the study and definition of different types of traceability relations; the generation of these relations; the development of architectures, tools, and environments for managing traceability information; and the empirical investigation of organisational practices regarding the establishment and deployment of traceability information in the software development life cycle.

MOTIVATION

A major characteristic of software engineering compared to other engineering disciplines, is our inability to get the product right the first time. One of the most reliable methods of ensuring problems, or failure, in a large, complex software project is to have poorly documented requirements specifications. The eliciting process is difficult by it self, but can be improved. Care should be taken to involve all of a project's significant stakeholders in the RE process. The stakeholders can be in-house or external personnel, and can include end-users, customer acceptance testers, customer contract officers, customer management, future software maintenance engineers, salespeople, etc. Anyone who later can derail the project, if their expectations are not met, should be included if possible. This work usually takes effort and is time consuming.

Other challenges can be defeated with less effort and even more to gain. First thing is to assure that the documented requirements are clear and understandable. Definition of complete, just enough detailed, and quantifiable requirements is a prerequisite to meet this challenge. Thus, choosing a requirement's quality-attributes in a precise and quantifiable way is important. Tool support can help structure and arrange the process, which can increase the efficiency of the process and raise the quality of the requirements. Further on, tool support can improve the accessibility, improve the outline and thereby clarify and help communicate the requirements. Light analysis to ensure consistency of the requirements can also be performed.

Change of requirements is an intrinsic and essential part of software development, and for good reasons. It is difficult to specify what is required without having some form of model to help formalize the problem. The real-world problem, which the software is intended to solve, is itself subject to change. And even if what to produce is known, the design and development of these invisible, intangible, and complex software products is still inherently difficult. This applies to both short and long-term software engineering; it is challenging to keep the requirements up to date with the system design and

source code and visa versa. In the same way it is challenging to trace and determine if a requirement is realized in a system. It may take serious effort to determine if an application has significant unexpected or hidden functionality which does not correspond to the requirements, and it can indicate deeper problems in the software development process. If the functionality is not necessary to the purpose of the application, it should be removed, as it may have unknown impacts or dependencies that were not taken into account by the designer or the customer. If not removed, information will be needed to determine risks and to determine any added testing needs or regression testing needs.

The challenge of validating that the actual functionality is in accordance with the requirements is also a standard aspect of projects that include Commercial Off-The-Shelf (COTS) software or modified COTS software. The COTS part of a project will typically have a large amount of functionality which is not included in project requirements, or may be simply undetermined.

A requirement management tool (RMT) with possibilities to define dependencies or relationships between requirements and other software artifacts from the development process can contribute to, and may provide functionality to, defeat these challenges mentioned. Relevant software artifacts can be design models, use case descriptions, source code and test definitions.

Such RMT may be able to perform automated in-depth analysis of software dependencies and reveal requirements and software artefact inconsistency. Ideally it will be able to reveal obsolete functionality –constituting a security threat, missing functionality –contributing to customer dissatisfaction or software failure, or what to change if requirements changes, which tests to run if anything changes, and so on.

Today, software has become a critical competitive factor for many organizations. The ability to produce software of high quality, for less cost, in shorter timescales is a matter of survival. Hence, organizations are forced to continually seek to improve their processes by which tools with support for quantification and traceability can make the difference.

PROBLEM DEFINITION

The aim of this project is to design and implement a requirement management tool with functionality for documenting quantifiable requirements. More specifically this means that the tool will:

- support organization requirements in categories and projects
- provide a basic,default set of requirements attributes
- support definition of customized requirements attributes

- support customised categorization of requirements with further support for attaching pre or user-defined requirement attributes as templates

The tool should provide support for simple traceability of requirements. More specifically simple traceability means that the tool will:

- support relating requirements to requirements
- support relating stakeholder to requirements
- provide an outline of the requirements
- provide functionality to display, compare and sort requirements in table view

Further, more advanced traceability techniques end technology could be explored.

The tool should be implemented as a client-server system, designed to support different client applications.

To obtain a complete picture of the research domain and to ensure taking the right design decisions, the most central concepts and methods in RE will be described. The most relevant and widespread existing tools will be described in brief.

Given the task of implementing a tool, the focus will be technical rather than focusing on the requirements engineering process in general.

PROJECT CONTEXT

This project is assigned by WesternGeco (WG) / Schlumberger [45] by Bjørn Nordmoen, Chief System Architect & Technical Advisor at the WG Oslo Technology Center. The purpose was originally to explore the possibilities of implementing a lightweight, easy-utilizing tool for requirement management, with support for quantification and traceability of requirements and appurtenant artifacts. The objective changed during the project and is now to deploy such an adequate tool. The tool will possibly be utilized in future software development projects at WG.

Further, the project is a part of the ModelWare IST research project [29], through WG's involvement in ModelWare. This has reached the press' attention; "Seismic company WesternGeco Schlumberger contributes to find methodologies and techniques to prevent source code of large IT-solutions obsoleting"[1] [9].

The ModelWare project is co-funded by the European Commission under the "Information Society Technologies" Sixth Framework Programme (2002-2006). The major stake of the ModelWare project is to increase European competitiveness in the domain

[1] Translated from Norwegian. Original text: "Seismikkselskapet WesternGeco Schlumberger bidrar til å finne metoder som hindrer at programkoden i større it-løsninger forfaller."

of software development. In that context there are two main actions leaded for ModelWare: the capitalization on the level of the technical domains, and the automation of the code generation. This project fits into the ModelWare project as requirement management tool contribution.

Related to the ModelWare project, a prototype of the tool of this project was demonstrated at the "European Conference on Model Driven Architecture - Foundations and Applications" [6], November 7-10th 2005, Nuremberg, Germany, by Bjørn Nordmoen.

At the current time the project resources is hosted at the author's private server `http://remato.eide.biz`. Compiled binaries of the ReMaTo client and source-code is found there. The server hosts an Eclipse update-site, '`http://remato.eide.biz/eclipse`', for automatically installing and updating the client as an Eclipse plug-in. The server is also running the ReMaTo server-service for test and demonstration purposes. This service is available at '`http://remato.eide.biz:8080/smartservice`'. A Java Web Start [18] edition of the tool is also available at the server at '`https://remato.eide.biz/jnlp/remato.jnlp`'.

In near future the source-code will be published as a part of the Model Driven Development integration project (MDDi) [28], at the SourceForge.net [41] or equivalent Open Source software development web site.

LIMITATION OF SCOPE

As mentioned in the Problem Definition, p. 3, the main goal of this project is to design and implement a requirement management tool. This implies a technical focus.

Due to the degree of technicality of the project, the studies of the requirement management process domain will be limited to a minimum, covering only those aspects and concepts necessary to explain the design and implementation choices. This limitation concerns particularly the eliciting-process of requirements.

The technical sides of the challenges will on the other hand be elaborated.

REPORT OUTLINE

This section presents a brief overview of the entire document, giving a short description of each part.

Introduction This chapter contains background information about the project, such as motivation, problem definition, project context, limitations of scope and this readers guide.

Project Process and Method This chapter describes the work process and methodologies used during the project. The most important tools used during this project is also described in brief.

Part I - Prestudy This part presents concepts related to RMTs.

- **Requirements Engineering** This chapter describes the process in which a RMT-tool can be utilized.
- **Requirements** This chapter describes the cornerstones of a RMT-tool.
- **Quantification** This chapter describes the concept quantification and strategies of achieving it.
- **Traceability** This chapter describes the concept traceability and techniques for achieving it.
- **Existing Tools** This chapter presents in brief existing tools on the market.

Part II - Contribution This part presents the main contributions of the project, which is the ReMaTo RMT.

- **Scenarios** This chapter describes in brief chosen scenarios.
- **Requirements** This chapter describes in brief requirements within the project's scope.
- **System Design** This chapter presents the system's overall architecture.
- **Implementation** This chapter presents details and examples of the actual implementation of latest release.
- **ReMaTo Editions** This chapter presents in brief the functionality implemented in latest release with screen prints.

Evaluation and Discussion This chapter elaborates on the choices made and evaluates the current state and project overall progress.

Conclusion and Further Work This part presents the report's conclusion and describes improvement and extension areas for the tool, as well as relevant topics for further research within the traceability domain.

PROJECT PROCESS AND METHOD

The chapter present an overview of the project's process, and tools used to design and implement the requirement management tool.

PROJECT PROGRESS

The methodology used in this project, was never formally discussed or chosen at the projects start. Given the requirements and the environments, it has more or less been formed during the projects course. The details are described in the following sections.

The initial overall work plan for the development of the tool and the writing of the report can be found in Appendix A.

Methodology

The nature of the project is explorative. Thus, a waterfall-based development process is inadequate as a life cycle model. For the core development process an agile, iterative approach has been used. The core development method can be categorized as a combination of modified Extreme Programming (XP) method and Agile Modeling (AM) [33].

The core development subprocess is illustrated as in figure 1. The rounded rectangles illustrate subprocesses and the arrows illustrate work flow, as in activity diagrams.

Due to scope of the projects resources, it can be categorized as an in-house-project, however with a COTS-perspective. Concerning the stakeholder roles, the assigner at WG represents customer, owner and end-user. The author has also been representing the end-user among others. The story cards of XP have been oral discussed and agreed upon among WG and the author. The prototypes has been developed, tested and evaluated rapidly.

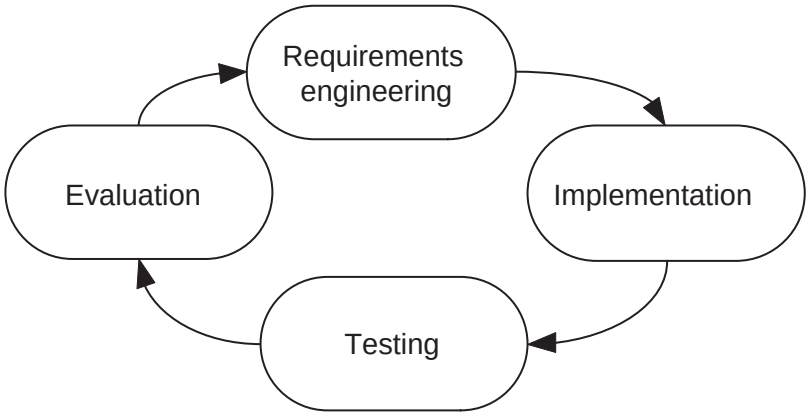

Figure 1: The core development process

Process

Figure 2 shows the overall layout of the project process. The skies illustrate superior subprocesses or phases and the rounded rectangles subprocesses. The arrows illustrate dependencies among phases and subprocesses, or work flow as in activity diagrams.

The project's process illustrated in figure 2 reflects the informal and agile approach to the development. After a brief initial exploration of the RE domain and exploration of existing RMTs, these activities have been performed in parallel with the prototype development iterations. In the projects course, there have been developed three main prototypes with large and clear distinctions. The differences have mainly been concerning architectural choices and strategies, while the supported functionality has been quite similar. Finally the report is written.

TOOLS

In order to perform the design and development iterations faster, this project has utilized software packages to support the work as much as possible.

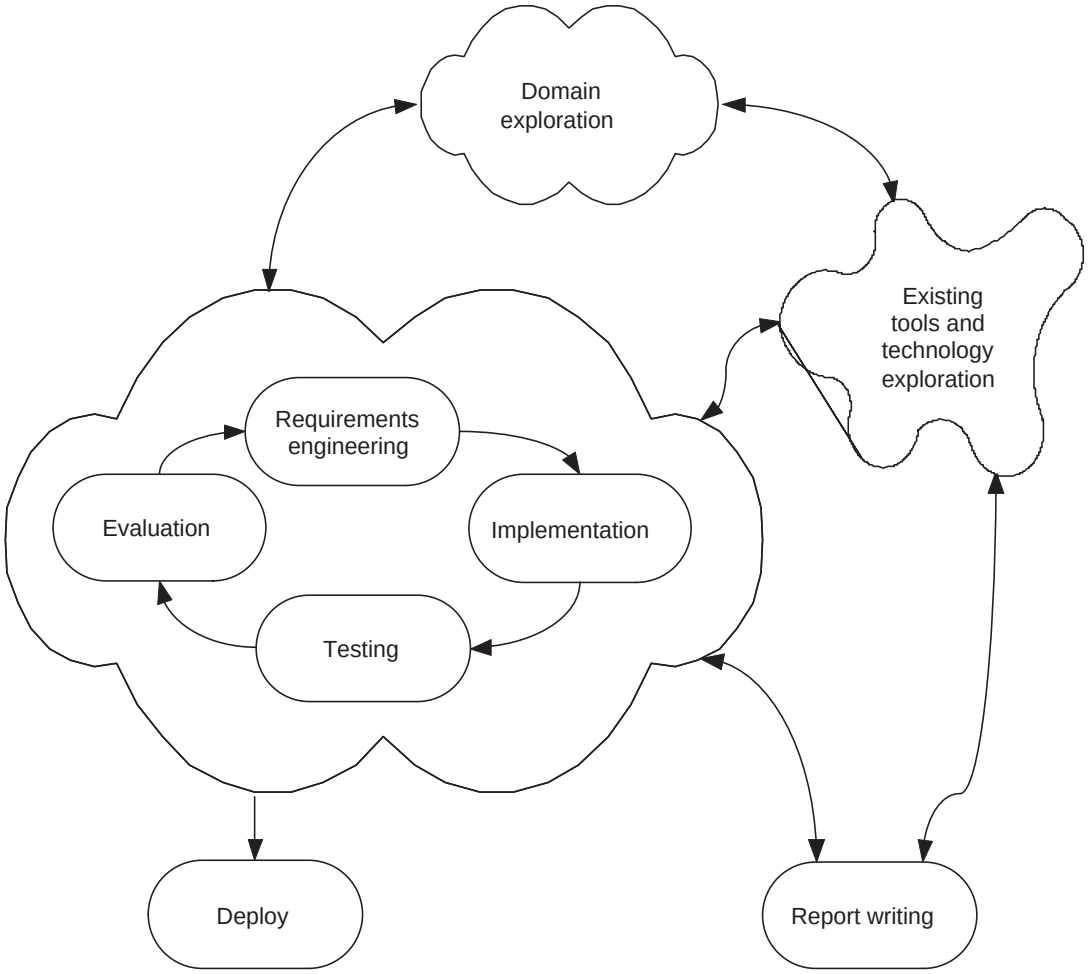

Figure 2: The overall project process

Eclipse

The Eclipse project [5] develops an open source, free Integrated Developer Environment (IDE), which is plug-in based and thus support several programming languages and compilers. Eclipse is a plug-in based IDE written in Java giving the benefit of using the same software independently of the workstation platform. Eclipse has built in support for Concurrent Versions System (CVS).

Borland Together Architect 2006

Borland Together Architect 2006 [2] (BTA) is part of Borland's Together software suite, providing developers with an efficient tool for modeling software using Unified Modeling Language (UML) diagrams and source code audits and metrics. BTA keeps the source code and UML diagrams consistent at all times and therefore gives an advantage when using an iterative design and implementation cycle. If decided to test aspects of the framework by writing code, the model is automatically updated to fit this and can be used to view the changes done during implementation.

TeXnicCenter

TeXnicCenter [43] is a feature rich integrated development environment (IDE) for developing LaTeX-documents in Microsoft Windows environment freely available under GPL.

Debian Linux Development Server

The project use a development server to run the server-service, make program available, host CVS repository, database services etc. Host: `http://remato.eide.biz`. The server possesses a OpenSSL-based, self-signed Root Certificate Authority (CA) used for encrypted communication and signing of Java ARchives (jar).

TPTP

The Eclipse Test and Performance Tools' Platform (TPTP) [44] is utilized to tune performance and verify quality of the system developed.

TestNG and JUnit

TestNG [42] and JUnit [20] is utilized to automatically test the server and the client-server communication.

Part I

Prestudy

Chapters

This part describes the main concepts of the project, quantification and traceability, with appurtenant topics. The appurtenant topics addressed are requirements and requirements engineering. In the end a selection of existing RMT tools are presented in brief. The purpose of this part is to get an overview of quantification and traceability of requirements' domain.

CHAPTER 1

REQUIREMENTS ENGINEERING

This chapter presents a brief overview of requirements engineering. The purpose is to describe the basics in order to later elaborate on quantification and traceability of requirements.

1.1 DEFINITION

Requirements engineering (RE) is an activity, which aim is to discover, document and maintain a set of requirements [26] [47]. The use of the term engineering implies that systematic and repeatable techniques should be used to ensure that system requirements are complete, consistent and relevant [40].

RE can be denoted either as a process or as an activity. In general, a process uses one or several methods, which consists of one or several activities. In other words, RE process refers to the complete set of activities, while RE-activity refers to a single activity or requirement.

Common activities related to RE are listed in table 1.1.

1.2 CONTEXT

There exists a broad spectrum of strategies or approaches for discovering or eliciting requirements in software engineering. The approach or RE process is described as a sequence of actions, during which the list of requirements for a new software system is elicited, analysed, validated and documented into a formal, complete and agreed requirements specification [39]. The best choice of methodology depends on the context in which the process takes place. The term context refers to the limited domain over

Table 1.1: Common activities related to RE

- Preliminary Risk Analysis
- Traceability Analysis
  - Software Requirements to System Requirements (and vice versa)
  - Software Requirements to Risk Analysis
- Description of User Characteristics
- Listing of Characteristics and Limitations of Primary and Secondary Memory
- Software Requirements Evaluation
- Software User Interface Requirements Analysis
- System Test Plan Generation
- Acceptance Test Plan Generation
- Ambiguity Review or Analysis

which a model or prediction applies [25][36]. This means that different contexts require different kind of attention and concerns. Further, the context influences the type of project desirable and how requirements are elicited and documented.

The contexts taxonomy has many dimensions. One dimension can be the type of projects [22][37] summarized in the table 1.2.

Table 1.2: Types of projects

- In-house development
- Product development (of COTS) / Vertical market situation
- Time-and-materials based development
- COTS purchase (no further assistance)
- COTS-based acquisition (w/ adaptation assistance)
- Tender (May be custom-built or COTS-based acquisition)
- Subcontracting
- Situation unknown (E.g., customer uncertain whether to buy COTS or have something custom-built)

The variety of situations in which software products and systems are developed, as listed in Table 1.2, reflects the variety of requirements documentation styles which is an advantage to master.

The different types of project contexts, from a developers point of view, are illustrated in Table 1.3. In a product development context, the specific users and customers are

unknown to the developer, while in a custom-built system project, they are usually both known to the developer.

Table 1.3: Context issues from developers point of view

customer \ user	known	unknown
unknown	??	product development
known	custom-build	client-product

The different types of project contexts from a customer's point of view are illustrated in Table 1.4. In a tender-based project the developer is usually unknown, while the user can be known or unknown. If both the developer and user are known to the customer there is also the distinction that customer and developer may be the same (persons, organization) or not.

Table 1.4: Context issues from customers point of view

developer \ user	known	unknown
unknown	tender	tender
known	custom-build	client-product

Another dimension to the RE context is illustrated in Figure 1.1. The spectrum of how different solutions are developed will also have an impact on the RE approach.

A fourth dimension contributing to determine the context is contract types and structures of projects [22]. Variants are illustrated in Table 1.5.

The initial analysis contract type includes requirements only, which is used in later tender or negotiation. Consultancies used in initial analysis may or may not compete in bid for later phases. Design and development projects with fixed price may be risky for the developers. Recovering loss in maintenance phase is a frequent tactic. Design and development projects with variable price are fairly seldom used, except for long-term customer relationships. All-in-one contracts are often preferred by customers with little IT expertise.

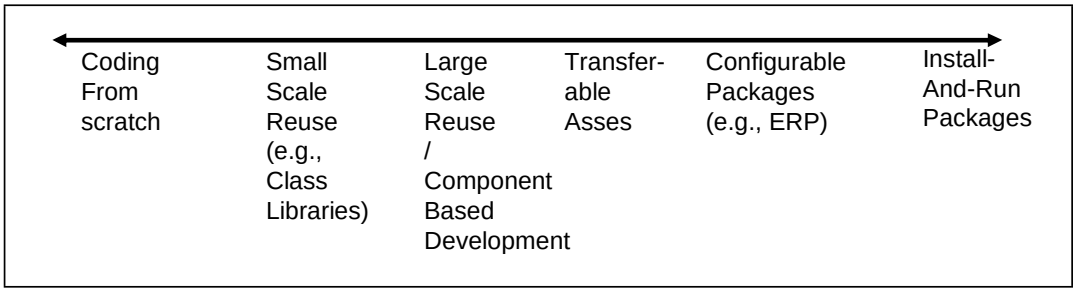

Figure 1.1: Various contexts for RE

Table 1.5: Contract types

- Initial analysis contract
- Design and development, fixed price
- Design and development, variable price
- All-in-one contract
- Development contract, fixed price

The last dimension presented here that influence RE context is the target level of requirements, often referred to as goal-design-scale [22]. Requirements differ from goals by being verifiable, while goals are not necessarily so. Organizations vary considerably in their target level and handling of requirements specifications. Different levels are listed in Table 1.6.

Table 1.6: Requirement levels (goal-design-scale)

- Goal level requirements
- Domain level requirements
- Design and development, variable price
- All-in-one contract

The goal level requirements state business objectives and are difficult to satisfy by software products alone. The next level, domain requirements, involves tasks of which software products can provide required support. The product level requirements are more detailed, specifying input and output. Finally, design level requirements are even more detailed, showing detailed interfaces and designed screen pictures. In some organizations requirements may end up in high level project plans, functional specification documents, design documents, or other documents at various levels of detail. No matter what they are called, some type of documentation with detailed requirements will be needed by the testers in order to properly plan and execute tests. Without such documentation, there will be no clear-cut way to determine if a software application is

performing correctly.

1.3 APPROACHES

The previous section describes the large variety in projects' context. This implies that a standard prescriptive approach will fail to identify the necessary and sufficient contents and style of a requirements document, because what is good enough in one situation may not be desirable or acceptable in another.

The first step of a general consideration is to be aware of the variance. Further, also to stress the importance of cooperative work when executing RE activities, in particular developing requirements through analyzing the problem, documenting the observations in a variety of representation forms, and checking the accuracy of the knowledge gained [24].

Being more specific, approaches to RE can be categorized in three main approaches [22] illustrated in Table 1.7.

Table 1.7: Three requirements approaches

- traditional approach (product-level requirements)
- fast approach (domain-level requirements)
- two-step (domain-level + design-level)

The traditional approach involves much elicitation from stakeholders like interviews, document analysis, workshops, brainstormings etc. The functional requirements are documented as function lists, feature requirements and textual process descriptions. This approach is well suited for COTS acquisition of technical products, sub-contracting and maintenance projects (enhancing a deployed system).

The fast approach implies more limited and focused elicitation from stakeholders with activities like discussions with expert users and studies of relevant documents. The functional requirements are documented as tasks to be done by the users and support required from the system. A weakness of this approach is the challenge to specify all functional requirements as user tasks, as some requirements may be hard to specify as this. This approach is well suited for most types of projects except the ones mentioned for traditional approach.

The two-step approach in Table 1.7 differentiates the level of the requirements documented. The first step elicits requirements on domain-level (Table 1.6). The next step documents chosen requirements on design level for the complex interfaces. This approach also includes user interfaces like screens and prototypes, and technical inter-

faces like communication formats and protocols. The domain level requirements can be useful for validation of design-level requirements and later maintenance.

CHAPTER 2

SOFTWARE REQUIREMENTS

This chapter presents a brief overview of the concept *software requirements*. The purpose is to describe the basics in order to later elaborate on quantification and traceability of requirements.

2.1 DEFINITION

The term requirement is defined differently by various authors. The Oxford dictionary [32] defines *requirement* as "that which is required' and further on the term *require* as having need, finding something necessary. Adapted to the software engineering domain, requirements can be refined as "a condition or capability needed by a user to solve a problem or achieve an objective" [23]. A requirement is categorized as either functional or non-functional. In this project both aspects of requirements are supported.

2.2 FUNCTIONAL REQUIREMENTS

Functional requirements specifies a function that a system or system component must be able to perform [15]. It can be documented in various ways. The most common ones are written descriptions in documents, and use cases. Use cases can be textual enumeration lists as well as diagrams, describing user actions.

2.3 NON FUNCTIONAL REQUIREMENTS

Non-functional requirements (NF) are any other requirement than functional requirements. They can be categorized as three types as listed in Table 2.1.

Table 2.1: NF types

• Data requirements • Constraints • Quality requirements

Data requirements are often grouped together with functional requirements in specifications and describe how functional requirements should be reflected in the system.

Constraints explicitly and intentionally restrict the system or process. A key property of a constraint is that a penalty or loss of some kind applies if the constraint is not respected. Constraints include limitations of the engineering process, systems or system components' functionality, or its life cycle.

Quality requirements describe wanted qualities of the product that are not directly related to functionality. Quality of software is important, hence requirements addressing quality is important, and lack of such is a frequent cause of projects failure. At the same time quality requirements can be hard to capture or elicit in contrast to functional requirements; Users, in interviews of discussion groups, talk intuitively about what they do in function, while quality is often implicit to domain knowledge and thus not talked intuitively about. Observations, and studies of similar systems, also tend to give more information about functionality than quality.

Different types of NF taxonomies are presented and described in detail in Section 3.2.

2.4 SOFTWARE REQUIREMENT SPECIFICATION

The term requirements specification or software requirements specification (SRS) is used in two ways in the literature. First, it is defined as the process or RE activity undertaken to specify requirements [39]. Second, the requirements specification is defined as a document, which contains a complete description of what the system should do without describing how it should do it [16]. The second meaning is also supported by the RE definition just presented. This report will use the latter interpretation of SRS, that is the requirements specification is the output of the RE process, where the requirements and requirements model are described using a formal language, and it is agreed upon among all the stakeholders.

The SRS is needed for many purposes; discussing customer needs including validation, knowing what to design and implement including verification, tracing and requirement management, estimations (such as cost and effort), contract negotiations and court cases.

A documented software requirement specification provides a baseline for both validation and verification. The software validation process cannot be completed without an established software requirements specification.

CHAPTER 3
QUANTIFICATION

> Not everything that can be counted counts, and not everything that counts can be counted.
>
> *–Albert Einstein*

This chapter presents the concept *quantification of requirements* and forms the basics for the choices made for the requirement management tool.

3.1 DEFINITION

The term quantification is most common used in logic theories, where it is defined as "a construct that specifies the extent of validity of a predicate, that is the extent to which a predicate holds over a range of things" [46]. The concept does however apply with highest relevance in RE, but has no common or widespread explicit definition related to this. To construct such a definition, an interpretation and decomposition of the term is needed.

A more general definition of quantification can be "determine the quantity of" or "measure or express as a quantity" [32]. Further, *quantity* is defined as "that property of things that is (in principle) measurable" [32]. Related to requirements this will mean to identify properties or attributes of requirements that are measurable. If this is supposed to give added value for projects, it will not be adequate to identify these properties for every requirement individually. Individually it will, as an example, not enable validation and analysis which quantification of requirements is supposed to. To gain the added values, categorizing a projects requirements in an appropriate amount of categories, which have the same type of properties, is needed. When the mutual, measurable properties of a category are identified, then the value of each properties of each

requirement is needed to be elicited. First then requirements can be properly, and possibly automatically, validated and analyzed, and the quantification of the requirement or requirement specification will be obtained.

Based on the previous paragraphs, the adapted definition of quantification to RE context, used further in this report, is declared as "identification of measurable, mutually properties of similar requirements and the elicitation of corresponding values for each requirement".

3.2 NF TAXONOMIES

The term *taxonomy* is defined as "the branch of science concerned with classification" or "scheme of classification" [32]. Related to NF requirements this means categories in which they can be classified.

There exist several taxonomies of NF today; the most common are listed in Table 3.1.

Table 3.1: NF taxonomies

- McCall & Matsumoto quality factors[27]
- ISO9126 [17]
- IEEE Std 830[16]
- VOLERE taxonomy [38]
- Firesmith taxonomy[8]

The taxonomies vary in complexity and detailedness. To give an example, two taxonomies will be presented.

The VOLERE taxonomy of NF requirements [38] is listed in Table 3.2.

Table 3.2: VOLERE taxonomy

- Look and feel: The spirit of the product's appearance
- Usability: Ease of use, accessibility, ease of learning the product
- Performance: How fast, how safe, how many, how accurate
- Operational: What operating environment the product must work in
- Maintainability and portability: Expected changes, time allowed to make them
- Security: Security and confidentiality of the product
- Cultural and political
- Legal: what laws and standards apply to the product

The Firesmith taxonomy [8] addresses quality requirements and is more detailed than the VOLERE taxonomy. It distinguishes between developer- and usage-oriented quality factors. The developer-oriented quality factors are listed in table 3.3.

Table 3.3: Firesmith developer-oriented taxonomy

- Maintainability
  - Correct-ability
  - Extensibility
- Portability
- Reuse-ability
- Scalability
- Verifiability
  - Testability

The usage-oriented quality factors are listed in table 3.4.

Different quality factors of taxonomies may contradict each other; robustness vs. performance, security vs. usability, security vs. interoperability, performance vs. portability etc.

Table 3.4: Firesmith usage-oriented taxonomy

- Audit-ability (enough records for financial audit)
- Branding (how well the brand name is exposed)
- Capacity (# things that can be handled)
- Configure-ability (how easily product can be reconfigured)
  - Internationalization (e.g., different countries, languages)
  - Personalization (personal user experience)
  - Subset-ability (easy to make different functionality subsets)
  - Variability (degree to which different variants exist)
- Correctness (product / outputs free from defects)
  - Accuracy (e.g., deviation in quantitative data)
  - Currency (e.g., data up to date)
  - Precision (dispersion of data, regardless of accuracy)
- Dependability (degree to which users can depend on the product), including:
  - Availability (e.g., minimal downtime)
  - Reliability (operates without failure under specified use)
  - Robustness (also functions under abnormal conditions)
    - Environmental tolerance
    - Error tolerance (wrong user input)
    - Failure tolerance (defect in system execution)
  - Safety (preventing / dealing with accidental harm)
  - Security (... malicious harm)
  - Survivability (degree to which essential / critical services continue to be provided in spite of accidental or malicious harm)
- Efficiency (consumption of resources, IT & human)
- Interoperability (properly interfaced with and working together with something else)
- Operational environment compatibility
- Performance
  - Jitter (time-precision of events)
  - Response time (time to initial response)
  - Latency (time to completion of task)
  - Schedule-ability (degree to which events can be scheduled to occur at planned times)
  - Throughput (# jobs that can be completed in a certain time period)
- Utility (degree to which prod can be accessed and used by various types of users)
  - Accessibility (e.g., handicapped users)
  - Install-ability (easy to install?)
  - Operability (possible to perform tasks in accordance with operations manual?)
  - Transportability (possible to move system physically? Works while moving? : mobility)
  - Usability (ease of use)
  - Withdraw-ability (degree to which a problematic version of the system can be withdrawn and replaced with a previous, working version, without problems for the users)

3.3 REQUIREMENT PROPERTIES

The different taxonomies described in section 3.2 form natural classification schemes for quantification as defined in section 3.1. A class within a taxonomy can be identified and described completely with a mutual set of properties. This is one of the main purposes of the taxonomy of requirements.

What then remains in order to be able to achieve quantifiable requirements, is to determine the properties that a requirement of a taxonomy should constitute of. To achieve quantification, the properties have to be measurable.

Gilbs' proposal for obtaining quantifiable requirements [11][12] aims at making each property consist of a set of six coherent attributes, listed in table 3.5

Table 3.5: Gibs attribute specification

- Scale
- Test
- Worst level
- Plan or goal level
- Best cause level
- Now level

The scale attribute defines the units of the target measure and the test how it is going to be measured along the scale, like volt and voltmeter. The worst limit is the worst acceptable case under any circumstances. The plan or goal level represents attainment of formal success. The best level is the best known achieved level anywhere, under any circumstances. The now level of the attribute is the current level in the system if any.

The "real" requirements in this property definition lie in worst level and plan level. The scale and test are meta-data and the rest a reference, setting the requirement property in perspective.

Further, it is possible to build hierarchies of such properties using the same structure as in a specific taxonomy, or building a new dimension across this.

3.4 BUSINESS APPLICATION

Using taxonomy templates with quantifiable properties, as described in section 3.3, can facilitate the elicitation and documentation process, by simply applied as checklists. Further, it will, as quantifying templates, contribute to achieve requirement quality, by

forcing definite documentation. Beyond this, it will by the predefined, mutually properties, enable validation and analysis, both manual and most important automatically by tool support.

CHAPTER 4
TRACEABILITY

Science has found that nothing can disappear without a trace.

–Dr. Werner von Braun, U.S. space program

This chapter presents the concept *traceability of requirements* and forms the basics for the choices made for the requirement management tool.

4.1 DEFINITION

Software traceability is the ability to relate artifacts which are created during the development of a software system (e.g., requirements, design and code artifacts) with each other, the stakeholders that created them, and/or the rationale underpinning their exact form [48].

Requirement traceability involves software traceability, but implies also specific traceability among requirements. A requirement-specific definition can be as the following; "the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases" [13]. This involves documenting the changes of a requirement, and making the history available. Related to this, it is also necessary to be able to define baselines. A baseline can be as simple as time-stamping all current requirements versions. This enables to later review the state of all requirements at a certain point in time.

The ability to search and select views of requirements, based on criteria for one or more properties of the requirements, is also a part of requirements traceability.

4.2 ARTIFACTS

The term "software artifacts" is well known from the project process methodology Rational Unified Process (RUP) [21] and can be any work product involved in the software life-cycle. In a traceability tool, artifacts are the basic elements. The most relevant artifact types to requirements traceability is listed in table 4.1.

Table 4.1: Artifact types

- other requirements
- stakeholders
- model elements
- code
- tests
- informal text artifacts (emails, chat sessions)

A more generic and technical approach to defining the artifacts is listed in table 4.2.

Table 4.2: Artifact level of granularity

1. Container
2. Concept
3. Concept property

The container is here the level that is managed and persisted (e.g. file, database). The concept is the actual artifact, as described above in table 4.1. The concept property can be a property of a requirement, property of a model element, font of a paragraph, etc.

4.3 TECHNIQUES

Traceability is achieved by defining and maintaining relationships to other artifacts created during system development [35] such as stakeholder needs, architectural or design elements, source code, to name a few.

4.3.1 Manual Techniques

Manual traceability techniques are characterized by occurring as lists or matrices in documents that are manually written and updated. The most common types of manual

traceability today [34] are listed in table 4.3.

Table 4.3: Common manual trace techniques

• Cross reference and indexing schemes • Traceability matrix

Traceability matrices are used to relate requirements to other software development artifacts. Usually requirements are listed along rows and the other artifacts along columns or visa versa. At each crossing of a column and a row a mark is made if the respective requirement and artifact are related. Different types of relationships can be indicated by using different marks.

Cross references and indexing schemes are references made across several artifacts to indicate links between them, or lists of indices containing the related artifacts for each one.

4.3.2 Automation Techniques

It is possible to automate traceability using the techniques described in section 4.3.1. However, using a traceability link concept is a more common strategy to achieve this.

A traceability link is a relationship that describes a traceability connection between specific artifacts. Such link-types are listed in table 4.4.

Table 4.4: Automated traceability link types

• Automatic Link
  – Derived link
  – Implied link
• Manual Link

An automatic link is characterized by being automatically created by a program. This can happen in at least two ways; Derived link can be computed from information about artifacts given a set of inference rules. Relationships in code are an example of derived traceability. Implied link can be computed from information about other links given a set of inference rules. Transitive closure is an example of implied traceability. A manual link is a kind of link that is created manually by a stakeholder. Relating a requirement to a source code package is an example of such link.

The real challenge for automation traceability is not to define the links and describe the links, but how to maintain them when the artifacts are changing. If different tools are used for developing the different artifacts, the complexity grows. More on these challenges is found in chapter 15.

4.4 BUSINESS APPLICATION

Traceability information can be used to support:

- The analysis of the implications and integration of changes requested in the system development process
- The maintenance and evolution of software systems and documentation
- The reuse of software systems and their components
- And the inspection and testing of software systems (verification).

Of these reasons, traceability of requirements has been recognised as a significant capability in the software development and maintenance process, and as an important factor for the quality of the final product.

A software requirements traceability analysis can also be conducted to trace software requirements to (and from) system requirements and to risk analysis results.

CHAPTER 5

EXISTING TOOLS

This chapter presents in brief the most common existing requirement management tools and mention others. Thoughts concerning a possible formal evaluation RMT tools is presented in the end of the chapter.

5.1 CALIBER RM

Borland CaliberRM 2005 is an enterprise requirements management system designed to facilitate collaboration, impact analysis, and communication in the definition and management of changing requirements. CaliberRM aims at all organizations - large, small, or distributed.

5.2 RATIONAL REQUISITEPRO

The IBM Rational RequisitePro solution is a requirements and use case management tool for project teams, designed to improve the communication of project goals, enhance collaborative development, reduce project risk and increase the quality of applications before deployment.

5.3 DOORS

DOORS (Dynamic Object Oriented Requirements System) is an Information Management and Traceability (IMT) tool to support RE process.

5.4 OTHER TOOLS

An extended list of existing tools is found in table 5.1.

Table 5.1: Existing tools list

- Caliber RM from Borland
- RequisitePro from Rational
- Doors from Telelogic
- IRqA from TCP Sistemas e Ingeniería
- OPHELIA an EC research project
- RDT from IGATECH Systems
- Reqtify from TNI Europe
- Themis/Papeete from Thales
- XTie-RT from Teledyne Technologies Incorporated
- Common word-processor

5.5 POSSIBLE EVALUATION

As described in previous sections, there exist various tools in the market. In is not in the project's scope to do a formal evaluation of the existing tools. However, if such an evaluation should be performed this section presents a suggestion for evaluation criteria.

The evaluation can cover six main areas:

- Quantification functionality
  - CRUD-functionality for requirements and stakeholders
  - Decomposition of requirement information
  - Defining userdefined properties
  - Define use case
- Traceability functionality
  - Requirement history
  - Relate requirements
  - Relate stakeholders to requirement

  - Relate various artifacts to requirement
- Userability
  - Easy getting started
  - Easy to adapt to different process methodologies
  - Easy and effective to use
- Interoperability or integration opportunities
  - Use of standardization
  - Integration with other tools
- Collaboration opportunities and accessibility
  - Client availability
  - Platform support
- Cost
  - Purchase cost (licence)
  - Maintenance cost

The quantification criteria are intended to reveal the opportunities for create, retrieve, update and delete (CRUD) requirements. Further, define and adjust the properties of a requirement to achieve quantifiable requirements. The traceability criteria are intended to reveal basic and advanced types of traceability support.

Part II

Contribution

Chapters

This part presents the requirement management tool ReMaTo, which is developed during this project. The scope of the presentations is in line with the projects scope. The purpose of this part is to enlighten the contributions of the project.

CHAPTER 6

SCENARIOS

This chapter presents in brief two example scenarios in line with projects scope, described in section "Problem Definition" (p. 3) and section "Limitation of Scope" (p. 5), and the development method described in chapter "Project Process and Method" (p. 7).

6.1 COLLABORATION OPPORTUNITIES AND ACCESSIBILITY

This scenario lets two stakeholders work on the same SRS by a RMT tool, each at different locations and different platforms.

At a software company's headquarter, a project leader is preparing a quarterly presentations for his current project's customers. In this work he is reviewing the status of the product's requirements in his requirement management tool on his Microsoft Windows platform. From the clear and detailed information available, he quickly gets the impression that the project is behind schedule, which surprises him. Hi picks up his phone to call John, a key developer of the project. On the phone, John, busy coding at the customers site, switches from Java Perspective to the RMT plug-in in his IDE. He can then inform that the latest updates have not been registered in the system. Five minutes after the phone call, John has updated the requirements, and can get back to coding again by switching view. At the headquarter the project manager exports the updated requirements as PDF-document and includes it in the customer's presentation. The customer is impressed over the control the manager tends to give.

6.2 QUANTIFICATION

This scenario lets the user build automatic tests from SRS defined in a RMT.

John, a developer, is about to begin verify the software maintaining after some comprehensive changes in the software's requirements. He switches to the RMT plug-in in his IDE to get an overview. He believes such changes are likely to occur frequently in the time to come, so he decides to write automatic tests based on the requirements. The tool support quantification by templates of properties. Thus, he is able write generic tests with little effort. Further, he is able to get information of expected software behaviour directly from the tool and into his tests. He runs the test, and is able to give a realistic estimate of resources needed to adjust the software to be up to date with the requirements.

A month later, the requirements are revised again. This time, John only selects a menu-button to execute his tests and is again able to give a realistic estimate of resources needed to adjust the software.

CHAPTER 7

REQUIREMENTS

This chapter presents in brief chosen requirements in line with the projects scope, described in "Problem Definition" (p. 3) and section "Limitation of Scope" (p. 5), and the development method described in chapter "Project Process and Method" (p. 7). A complete SRS for the system, written by Bjørn Nordmoen, is attached in appendix B, p. 104.

7.1 QUANTIFICATION

F-Q1 The system must provide CRUD functionality for requirements. This is the core functionality of the tool.

F-Q2 The system must provide functionality to support a hierarchical project and categorization schema. Multiple root-projects with sub-projects must be supported. A project must be able to contain one or more categories. A category must be able to contain one or more subcategories or one or more requirements.

This will provide support for defining customized, hierarchical NF taxonomies as described in section 3.2.

F-Q3 A requirement must contain a default set of relevant properties, as specified in the detailed requirements, p. 104.

F-Q4 The system must support CRUD for properties templates as described in section 3.3.

F-Q5 Further the system must support to relate the templates of F-Q4 to categories.

Together with the customizable category hierarchy, this will provide complete quantification functionality for requirements. Simultaneously it arranges for reuse of the templates, within a project, or among several projects.

7.2 SIMPLE TRACEABILITY

F-T1 The system must provide functionality to relate requirements to each other.

F-T2 The system must provide CRUD functionality for stakeholders.

F-T3 The system must provide functionality for relating stakeholders to requirements, denoted as a role.

7.3 ACCESSIBILITY

NF-A1 The system must be accessible from most common platforms, which means Microsoft, *nix and Apple OS-X.

NF-A2 The system must be accessible to developers in their environment as plug-in to the most common IDEs.

NF-A3 The system must be accessible to non-technical stakeholders.

7.4 COLLABORATION

NF-C1 The must be available from any location connected to the internet.

NF-C2 The system should handle simultaneously users.

CHAPTER 8

SYSTEM DESIGN

This chapter present an overview of the system design.

8.1 HIGH-LEVEL ARCHITECTURE

Too meet NF-C1, the system-design has adapted a client-server pattern. This arrange for users to use the system wherever connected to the internet. The high-level architecture is illustrated in figure 8.1.

The client is implemented as an Eclipse plug-in [5] reusing the GUI-components provided by Eclipse. Further, the client communicates with the server using Hessian [14], which implements RPC over the HTTP protocol. The server is implemented as a `Servlet` extending the `HessianServlet`. The server will handle the persistent data through the object relational mapping-layer (ORM) Hibernate. Hibernate can utilize most known SQL-based physical databases. Hibernate facilitates the database schema generation and data mapping between relational tables and the object-oriented domain model.

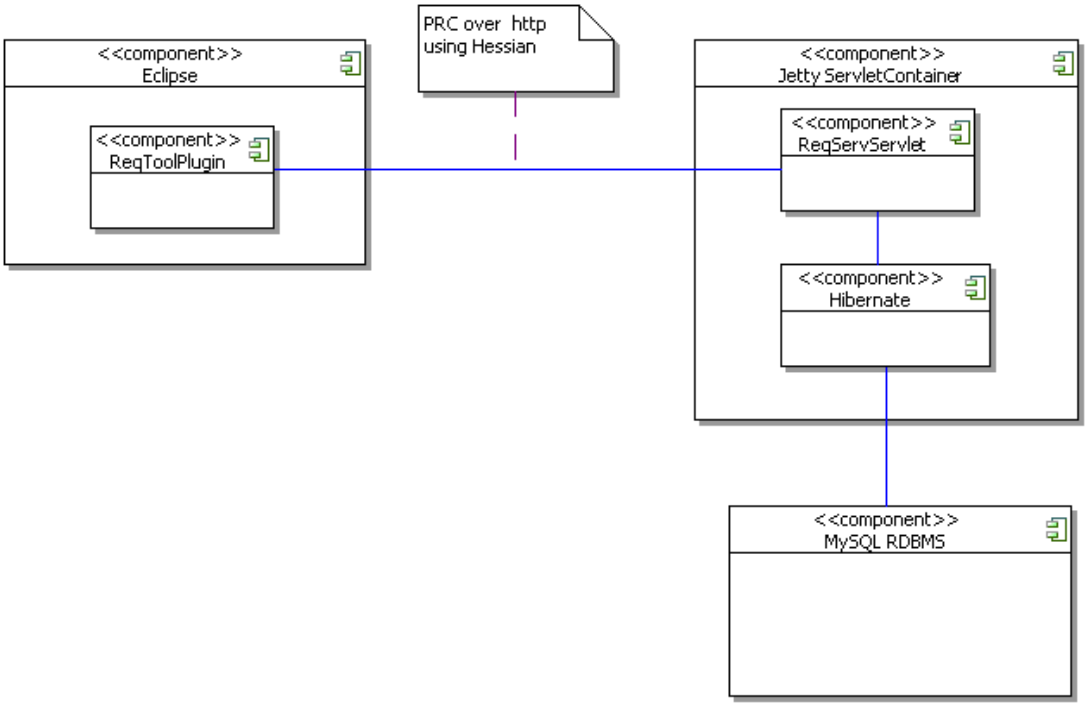

Figure 8.1: High-Level Architecture

8.2 DOMAIN MODEL

Evans' principles of domain-driven design [7] form the foundation for the system's domain model. The domain-model is presented in figure 8.2 using UML [10].

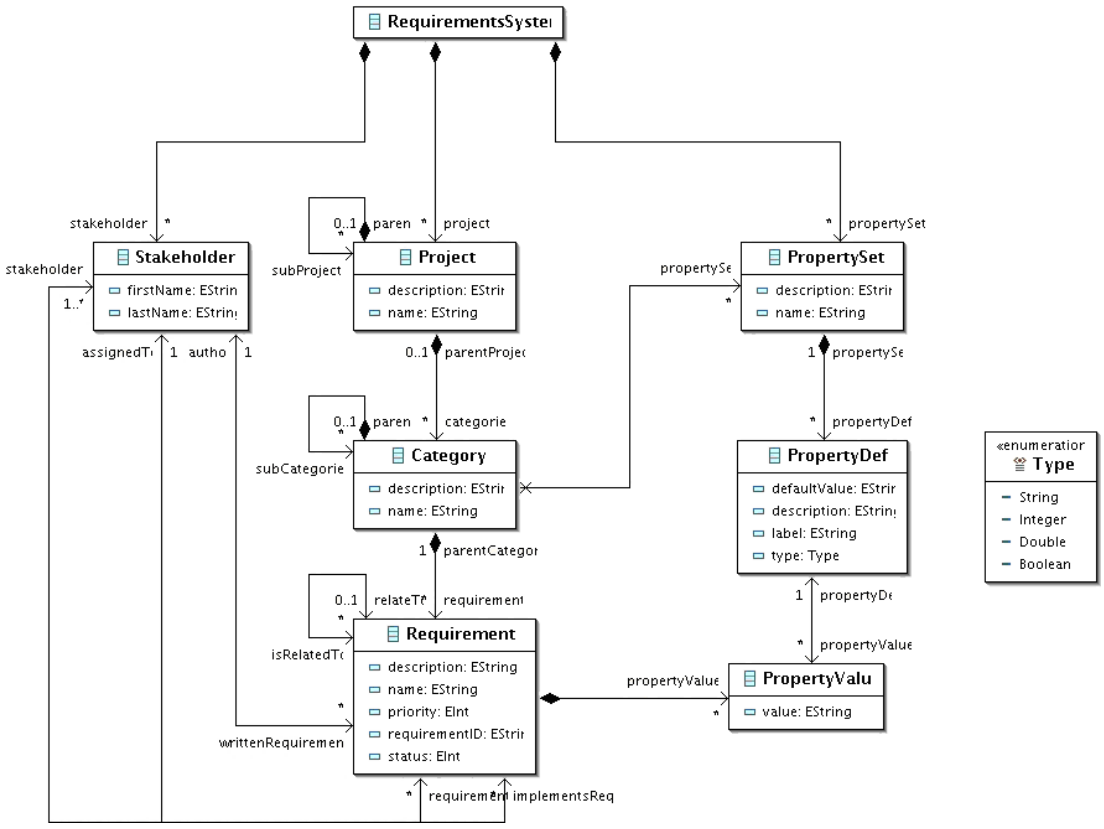

Figure 8.2: Domain Model as UML

The domain model arrange for a solution that meets the requirements described in chapter 7. As illustrated, both project and category support hierarchical nesting. The user-defined property templates are supported through the `PropertySet` type, which contains property definitions (`PropertyDef`). `PropertySet` can be connected to `Category`, which will imply that requirement of the respective category will acquire corresponding values (`PropertyValue`) as the property templates' property definitions. Thus, `PropertyValue` is the instantiation of a `PropertyDef` for a specific requirement, containing a specific value.

CHAPTER 9

IMPLEMENTATION

This chapter presents the actual implementation of the system.

The overall package structure is as the following:

- `remato.client`
- `remato.common`
- `remato.server`

9.1 THE `remato.common` PACKAGE

The common package consists of the source code shared by client and server. It is compiled as a separated jar-file before deployed with client and server. Figure 9.1 gives an overview of the package.

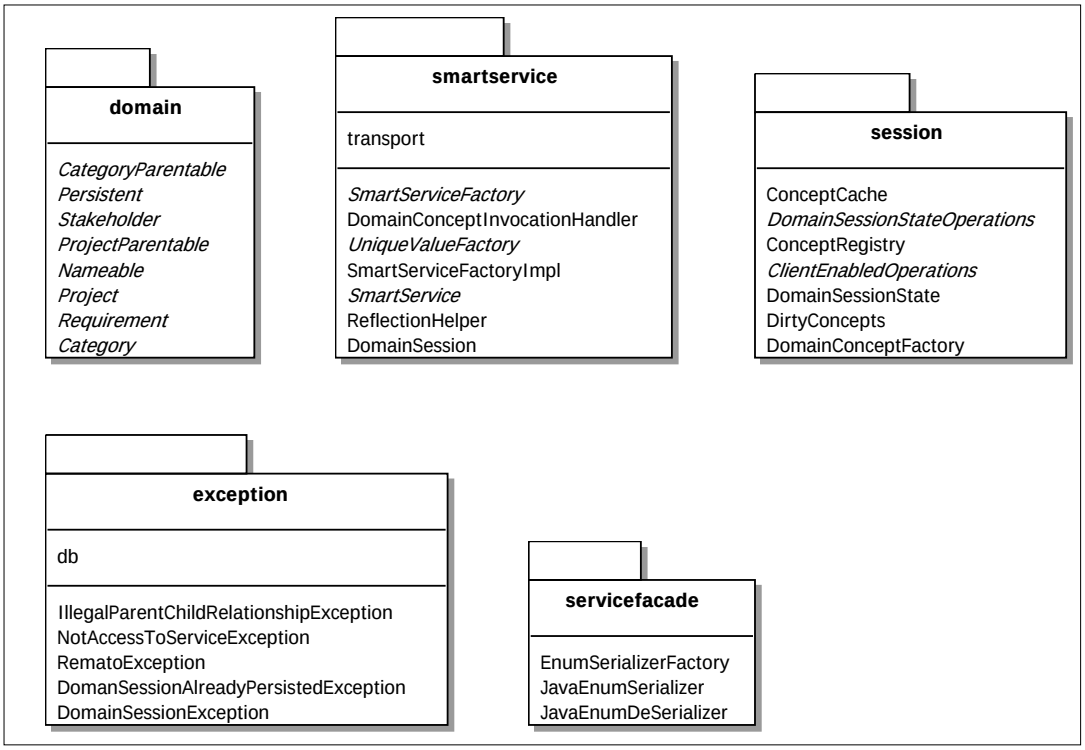

Figure 9.1: The `remato.common` package

As figure 9.1 illustrates, the domain model is located in `remato.common`. The domain model package contains "plain old java objects" (POJO). The implementation is illustrated in figure 9.2.

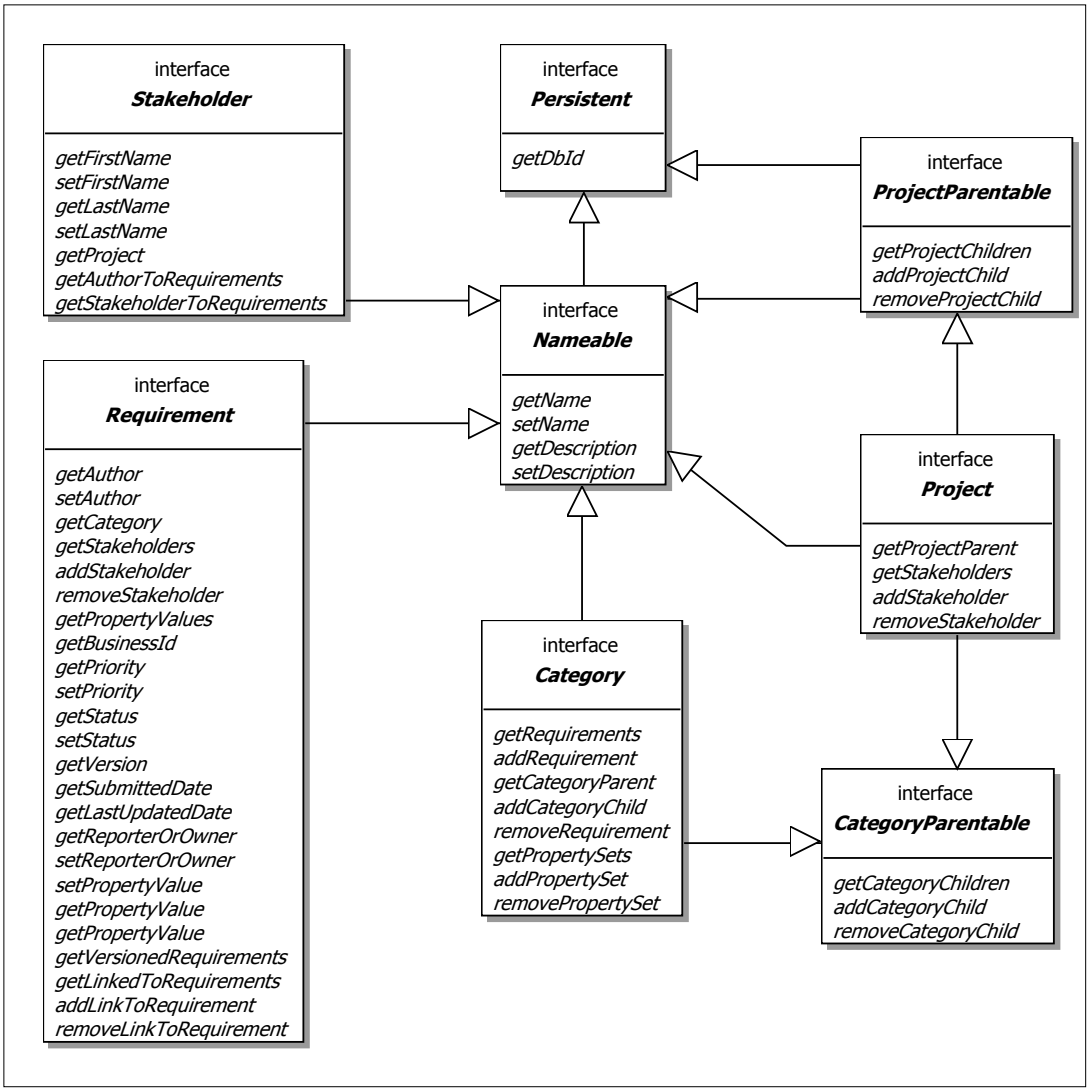

Figure 9.2: Selected interfaces from the `remato.common.domain` package

The package `remato.common.smartservice` contains the facade to the server. The main interface to the server is presented in listing 9.1.

Listing 9.1: Interface `remato.common.smartservice.SmartService.java`

```
1  package remato.common.smartservice;
2
3  import java.util.Collection;
4
5  import remato.common.domain.Persistent;
6  import remato.common.exception.RematoException;
7  import remato.common.exception.db.DatabaseException;
8  import remato.common.exception.db.InvalidDatabaseEntryException;
9  import remato.common.exception.db.MissingResourceException;
10 import remato.common.smartservice.transport.RetrieveRequest;
11 import remato.common.smartservice.transport.SaveRequest;
12 import remato.common.smartservice.transport.SaveResponse;
13
14
15 public interface SmartService extends UniqueValueFactory {
16
17   public void createDbSchema()
18   throws RematoException, DatabaseException;
19
20   public SaveResponse save(SaveRequest saveRequest)
21   throws RematoException, DatabaseException,
       InvalidDatabaseEntryException;
22
23   public Collection<Persistent> retrieveCollection(RetrieveRequest
       retrieveRequest)
24   throws RematoException, DatabaseException,
       MissingResourceException;
25
26   public Persistent retrievePersistent(RetrieveRequest
       retrieveRequest)
27   throws RematoException, DatabaseException,
       MissingResourceException;
28
29 }
```

The package `remato.common.session`, illustrated in appendix C, figure C.1, consists of a proxy which provides seamless connectivity with the server. The proxy loads objects lazy form server when needed, decreasing the probability for conflicts if simultaneously users connected to the server. This addresses NF-C2 in section 7.4. The package `remato.common.session` is not needed by the server. Still it is placed in `remato.common` to be available to other potential client implementations.

The clients can however access the interface presented in listings 9.1 directly, if wanted or needed.

9.1.1 The `remato.common` metrics

To give a brief overview of the `remato.common` package without including to many class diagrams, some metrics are listed in table 9.1. Explanation to the columns is documented in the glossary, chapter C.1, page 113. The metrics are computed by Borland Together Architect.

Table 9.1: Metrics of remato.common

Resource	**LOC**	**NOC**	**NOIS**	**NOM**	**NOO**	**PIS**	**PS**
common	3101	64	41	50	36		
remato.common.domain	105	10	4	24	24	10	10
remato.common.domain.impl	681	11	10	48	35	8	11
remato.common.domain.impl.base	29	2	1	6	4	2	2
remato.common.domain.property	23	4	3	5	5	4	4
remato.common.exception	130	5	1	1	1	2	5
remato.common.exception.db	181	9	1	0	0	0	9
remato.common.servicefacade	85	3	6	4	2	1	3
remato.common.session	799	7	18	31	25	1	7
remato.common.smartservice	798	7	41	50	36	5	7
remato.common.smartservice.transport	270	6	6	15	10	6	6

In brief; LOC means “lines of code” and NOC means “number of classes”.

9.2 THE remato.client PACKAGE

The remato.client consists of the client specific source code. The Eclipse plug-in skeleton classes is implemented and located here. Further, necessary parts of the Eclipse specific API is employed to obtain seamlessly integration with the IDE. The Model View Controller (MVC) is applied.

Figure 9.3 gives an overview of the package.

Figure 9.3: The remato.client package

In order to optimize the GUI implementation, the client needs more functionality in the model than the domain model objects provide. However, it is not desirable put such client specific code in the domain model, which is shared with the server and possibly other clients. To avoid the dilemma, the author has made a wrapper model to

the domain. It is adjusted to client needs while, wrapping around the domain model objects.

An example of such wrapper is illustrated in listing 9.2.

Listing 9.2: Class `remato.client.model.PropertyDefinitionSetWrapper.java`

```
1  package remato.client.model;
2
3  import java.util.ArrayList;
4  import java.util.Collection;
5  import java.util.Collections;
6  import java.util.Hashtable;
7  import java.util.LinkedList;
8  import java.util.List;
9  import java.util.Map;
10
11 import org.eclipse.swt.graphics.Image;
12 import org.eclipse.swt.layout.FillLayout;
13 import org.eclipse.swt.widgets.Composite;
14 import org.eclipse.ui.forms.IManagedForm;
15 import org.eclipse.ui.views.properties.IPropertySource;
16
17 import remato.client.controller.IRequirementPluginImages;
18 import remato.client.controller.RequirementPlugin;
19 import remato.client.view.editor.IModelFormPart;
20 import remato.client.view.editor.properties.PropertiesFormPage;
21 import remato.client.view.editor.usertypes.UsertypesSectionPart;
22 import remato.client.view.util.ModelItemPropertySource;
23 import remato.common.domain.Nameable;
24 import remato.common.domain.Persistent;
25 import remato.common.domain.RequirementSystem;
26 import remato.common.domain.property.PropertyDef;
27 import remato.common.domain.property.PropertySet;
28 import remato.common.smartservice.DomainConceptInvocationHandler;
29
30 /**
31  * @author petterei
32  *
33  * @version $Id: PropertyDefinitionSetWrapper.java,v 1.11 2005-12-19
        13:34:01 petterei Exp $
34  */
35 public class PropertyDefinitionSetWrapper extends NodeChild<
      ProjectRootSystemWrapper> implements IPropertyDefinitionContainer
       {
36   private PropertySet propertySet;
37   private Map<PropertyDef, PropertyDefinitionWrapper>
        propertyDefinitions;
38   private List<PropertyDefinitionWrapper> sortedPropertyDefinitions;
39   private static PropertySpecification<Persistent> lastUpdatedDate;
40
41   static {
```

```
42      lastUpdatedDate = new PropertySpecification<Persistent>(100, "
            Last update", PropertyType.DATE) {
43        @Override
44        protected Object getValue(Persistent persistent) {
45          return ((PropertySet)persistent).getUpdatedDate();
46        }
47        @Override
48        protected void setValue(Persistent item, Object value) {}
49      };
50      lastUpdatedDate.setDescription("Date and time of last update of
            the propertySet.");
51      lastUpdatedDate.setPosition(100);
52      lastUpdatedDate.setDefaultValue("0000.00.00 00:00");
53      lastUpdatedDate.setReadOnly(true);
54    }
55
56    /**
57     * Use when created from gui, then set parent.
58     */
59    public PropertyDefinitionSetWrapper(PropertySet propertySet) {
60      super(propertySet);
61      init(propertySet);
62    }
63
64    /**
65     * Use when created from server.
66     */
67    protected PropertyDefinitionSetWrapper(PropertySet propertySet,
          ProjectRootSystemWrapper wrapper) {
68      super(propertySet,wrapper);
69      wrapper.addDefinitionSet(this);
70    }
71
72    protected void init(Nameable nameable){
73      this.propertySet = (PropertySet)nameable;
74      propertyDefinitions = new Hashtable<PropertyDef,
            PropertyDefinitionWrapper>();
75      sortedPropertyDefinitions = Collections.synchronizedList(new
            LinkedList<PropertyDefinitionWrapper>());
76      initPropertyFromDomainObject(lastUpdatedDate);
77    }
78
79    /**
80     * @see remato.client.model.ClientModelItem#getSortingOrder()
81     */
82    @Override
83    public int getSortingOrder() {
84      return 10;
85    }
86
87    /**
```

```
88      * @see remato.client.model.IPropertyDefinitionContainer#
            getPopertyDefinitions()
89      */
90     public synchronized Collection<PropertyDefinitionWrapper>
           getPopertyDefinitions() {
91       for (PropertyDef def : propertySet.getPropertyDefs()) {
92         if (! propertyDefinitions.containsKey(def)) {
93           PropertyDefinitionWrapper spec = new
                PropertyDefinitionWrapper(def);
94           propertyDefinitions.put(def, spec);
95           int index = ((List)propertySet.getPropertyDefs()).indexOf(
                def);
96           sortedPropertyDefinitions.add(index,spec);
97         }
98       }
99       return sortedPropertyDefinitions;
100    }
101
102    public Collection<AbstractPropertySpesification>
           getAbstractPopertyDefinitions() {
103      ArrayList<AbstractPropertySpesification> returnList = new
             ArrayList<AbstractPropertySpesification>();
104      returnList.addAll(getPopertyDefinitions());
105      return returnList;
106    }
107
108    public synchronized void addPropertyDefinition(
           PropertyDefinitionWrapper definition) {
109      propertySet.addPropertyDef(definition.getDomainModelDefinition()
             );
110      propertyDefinitions.put(definition.getDomainModelDefinition(),
             definition);
111      sortedPropertyDefinitions.add(definition);
112    }
113
114    public synchronized void removePropertyDefinition(
           PropertyDefinitionWrapper definition) {
115      propertySet.removePropertyDef(definition.
             getDomainModelDefinition());
116      propertyDefinitions.remove(definition.getDomainModelDefinition()
             );
117      sortedPropertyDefinitions.remove(definition);
118    }
119
120    public synchronized void updatePropertyDefinition(
           PropertyDefinitionWrapper element) {
121      int position = sortedPropertyDefinitions.indexOf(element);
122      sortedPropertyDefinitions.remove(element);
123      sortedPropertyDefinitions.add(position, element);
124    }
125
```

```
126   public synchronized boolean insertAfter(PropertyDefinitionWrapper
          insert, PropertyDefinitionWrapper after) {
127     if (insert.equals(after)) {
128       return false;
129     }
130     sortedPropertyDefinitions.remove(insert);
131     int newIndex = sortedPropertyDefinitions.indexOf(after) + 1;
132     sortedPropertyDefinitions.add(newIndex, insert);
133     int position = sortedPropertyDefinitions.get(newIndex -1 ).
            getPosition() + 1;
134     for (PropertyDefinitionWrapper spec : sortedPropertyDefinitions.
            subList(newIndex, sortedPropertyDefinitions.size())) {
135       spec.setPosition(position);
136       position++;
137     }
138     return true;
139   }
140
141   @Override
142   public synchronized void setParent(ProjectRootSystemWrapper
          newParent) {
143     if (newParent != null) {
144       if (DomainConceptInvocationHandler.getActualConcept(newParent.
              getDomainModelObject()) instanceof RequirementSystem) {
145         ((RequirementSystem)newParent.getDomainModelObject()).
                addPropertySet(propertySet);
146         super.setParent(newParent);
147         if (null != getParent() && getParent() instanceof
                ProjectRootSystemWrapper && ! getParent().equals(
                newParent)) {
148           getParent().removeDefinitionSet(this);
149         }
150         newParent.addDefinitionSet(this);
151       }
152     } else if (null == newParent) {
153       getParent().removeDefinitionSet(this);
154       ((RequirementSystem)getParent().getDomainModelObject()).
              removePropertySet(propertySet);
155       super.setParent(newParent);
156     }
157   }
158
159   /**
160    * @see remato.client.model.ClientModelItem#getPropertiesFormParts
           (remato.client.view.editor.properties.PropertiesFormPage, org.
           eclipse.ui.forms.IManagedForm)
161    */
162   @Override
163   public List<IModelFormPart> getPropertiesFormParts(
          PropertiesFormPage page, IManagedForm managedForm) {
164     List<IModelFormPart> formParts = super.getPropertiesFormParts(
```

```
        page, managedForm);
165     Composite body = managedForm.getForm().getBody();
166     Composite section = managedForm.getToolkit().createComposite(
            body);
167     section.setLayout(new FillLayout());
168     formParts.add(new UsertypesSectionPart(page, section));
169     return formParts;
170   }
171
172   @Override
173   public Object getAdapter(Class adapter) {
174     if (adapter == IPropertySource.class) {
175       ModelItemPropertySource propertySource = new
              ModelItemPropertySource(this);
176       propertySource.addAbstractPropertySpesification(
              lastUpdatedDate);
177       return propertySource;
178     }
179     return super.getAdapter(adapter);
180   }
181
182   @Override
183   public Image getImage() {
184     return RequirementPlugin.getImageDescriptor(
            IRequirementPluginImages.IMG_PROPERTY_SET).createImage();
185   }
186
187 }
```

As the listing illustrates, the domain-model-object is set in the wrapper constructor and all appurtenant methods forwards to the domain model instance. Another feature of the client model is that no wrappers have hard-coded attributes in the same way as the domain model. The GUI model is more generic and utilizes only 'PropertyDefinition-Wrapper' and corresponding 'PropertyValue' which wraps to most of the plain getter and setter methods in the domain model objects. The reason of this is to be able to treat the objects more uniformly in the GUI tier. These attribute wrappers are defined static for each class they are needed for.

9.2.1 The `remato.client` metrics

Due to complex Eclipse class libraries used in the client, or possible bug in Borland Together Architect, BTA was not able to compute metrics for the client package. Another, less advanced, Eclipse plug-in, VisCount, produced the results of table 9.2.

Table 9.2: Metrics of remato.client

Benchmark	File count	Local class count	Line count
`client`	99	46	16,985

9.3 THE `remato.server` PACKAGE

As illustrated in figure 8.1, the server provides a servlet container and a database interface. The diagram of the main packages is illustrated in figure 9.4.

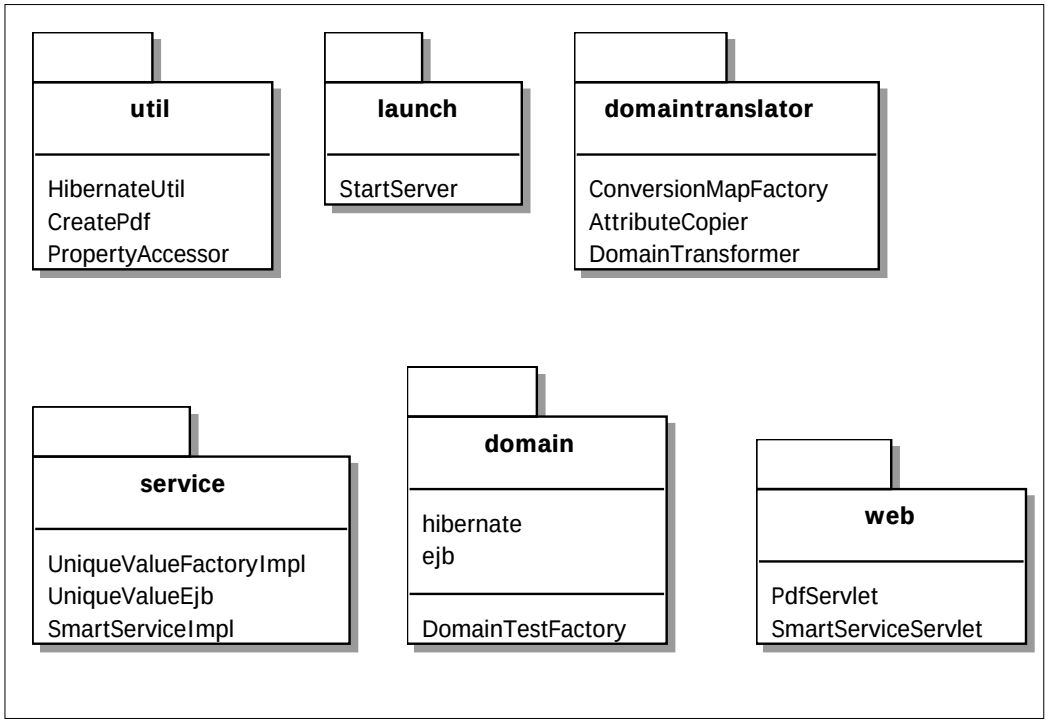

Figure 9.4: The `remato.server` package

To utilize Hibernate in the most maintainable way, modification of the domain objects is also needed at the server. Unfortunately, the wrapper-pattern applied at the client, is not compatible with Hibernate. Hibernate uses a CGLib-enhanced objects [3] which causes this lack of interoperability; it does not operate with "real" instances. Hibernate does however support to send the "CGLib-proxies" to the client, but that is not desirable for the same reasons as why the domain model was not modified at the client. A transformation is therefore needed. By implementing the model-interfaces and extending the objects in the domain model, consistency in the source code is preserved to some degree. An example of how this is done is presented in listing 9.3.

Listing 9.3: Class `remato.server2.domain.ejb.RequirementEjb.java`

```
1  package remato.server2.domain.ejb;
2
3  import java.util.Date;
4  import java.util.Set;
5
6  import javax.persistence.Basic;
7  import javax.persistence.CascadeType;
8  import javax.persistence.Entity;
9  import javax.persistence.GeneratorType;
10 import javax.persistence.Id;
11 import javax.persistence.Inheritance;
12 import javax.persistence.InheritanceType;
13 import javax.persistence.JoinColumn;
14 import javax.persistence.JoinTable;
15 import javax.persistence.ManyToMany;
16 import javax.persistence.ManyToOne;
17 import javax.persistence.OneToMany;
18 import javax.persistence.Table;
19 import javax.persistence.TemporalType;
20
21 import remato.common.domain.Category;
22 import remato.common.domain.Requirement;
23 import remato.common.domain.Stakeholder;
24 import remato.common.domain.VersionedRequirement;
25 import remato.common.domain.impl.RequirementImpl;
26 import remato.common.domain.property.PropertyValue;
27
28 @Entity
29 @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
30 public class RequirementEjb extends RequirementImpl {
31
32
33   @Override
34   @Id(generate = GeneratorType.AUTO)
35   public long getDbId() {
36     return super.getDbId();
37   }
38
39   @Override
40   public String getDescription() {
41     return super.getDescription();
42   }
43
44   @Override
45   public String getName() {
46     return super.getName();
47   }
48
49   @Override
50   @ManyToOne(targetEntity = CategoryEjb.class)
```

```
51    public Category getCategory() {
52      return super.getCategory();
53    }
54
55    @Override
56    @ManyToOne(targetEntity = StakeholderEjb.class)
57    public Stakeholder getPersistableAssignedTo() {
58      return super.getPersistableAssignedTo();
59    }
60    @Override
61    @ManyToOne(targetEntity = StakeholderEjb.class)
62    public Stakeholder getPersistableReporterOrOwner() {
63      return super.getPersistableReporterOrOwner();
64    }
65
66    @Override
67    @ManyToMany(targetEntity = StakeholderEjb.class, cascade = {
          CascadeType.ALL })
68    @JoinTable(table = @Table(name = "requirementStakeholder"),
          joinColumns = { @JoinColumn(name = "requirementDbId") },
          inverseJoinColumns = { @JoinColumn(name = "stakeholderDbId") })
69    public Set<Stakeholder> getStakeholders() {
70      return super.getStakeholders();
71    }
72
73    @Override
74    @OneToMany(targetEntity = PropertyValueEjb.class, cascade = {
          CascadeType.ALL })
75    @JoinColumn(name="requirement")
76    public Set<PropertyValue> getPropertyValues() {
77      return super.getPropertyValues();
78    }
79
80    @Override
81    @OneToMany(mappedBy = "currentRequirement", targetEntity =
          VersionedRequirementEjb.class, cascade = {
82        CascadeType.PERSIST, CascadeType.MERGE, CascadeType.REFRESH })
83    public Set<VersionedRequirement> getVersionedRequirements() {
84      return super.getVersionedRequirements();
85    }
86
87    @Override
88    @ManyToMany(targetEntity = RequirementEjb.class, cascade = {
          CascadeType.PERSIST, CascadeType.MERGE })
89    @JoinTable(table = @Table(name = "LINKED_REQUIREMENT"),
          joinColumns = { @JoinColumn(name = "LINK_FROM_ID") },
          inverseJoinColumns = { @JoinColumn(name = "LINK_TO_ID") })
90    public Set<Requirement> getLinkedToRequirements() {
91      return super.getLinkedToRequirements();
92    }
93
```

```
94    public VersionedRequirementEjb addVersionedRequirement(){
95      VersionedRequirementEjb versionedRequirement = new
            VersionedRequirementEjb();
96      super.addVersionedRequirement(versionedRequirement);
97      return versionedRequirement;
98    }
99
100   @Override
101   public Long getBusinessId() {
102     return super.getBusinessId();
103   }
104
105   @Override
106   @Basic(temporalType=TemporalType.TIMESTAMP)
107   public Date getLastUpdatedDate() {
108     return super.getLastUpdatedDate();
109   }
110
111   @Override
112   public Integer getPriority() {
113     return super.getPriority();
114   }
115
116   @Override
117   public Integer getStatus() {
118     return super.getStatus();
119   }
120
121   @Override
122   @Basic(temporalType=TemporalType.TIMESTAMP)
123   public Date getSubmittedDate() {
124     return super.getSubmittedDate();
125   }
126
127   @Override
128   public String getVersion() {
129     return super.getVersion();
130   }
131
132
133 }
```

As the listing illustrates, the constraints of the fields' relationships are expressed as annotations at the getter methods. In addition to facilitate retrieval and saving to persistent store, Hibernate initialized the database by SQL-schema export.

9.3.1 The `remato.server` metrics

To give an brief overview of the `remato.server` without showing to many class diagrams, chosen metrics are included in table 9.3. Explanation to the columns is documented in the glossary, chapter C.1, page 113. The metrics are computed by Borland Together Architect.

Table 9.3: Metrics of remato.server

Resource	**LOC**	**NOC**	**NOIS**	**NOM**	**NOO**	**PIS**	**PS**
server2	2397	21	27	22	16		
remato.server2.domain.ejb	536	9	24	16	16	0	9
remato.server2.domaintranslator	377	3	13	18	14	2	3
remato.server2.launch	149	1	16	15	7	0	1
remato.server2.service	340	3	27	16	15	0	3
remato.server2.util	449	3	22	22	16	2	3
remato.server2.web	546	2	24	21	15	0	2

In brief; LOC means “lines of code” and NOC means “number of classes”.

CHAPTER 10

REMATO EDITIONS

This chapter presents the results of the ReMaTo tool in screen shots and brief descriptions of the pertaining functionality. The tool is deployed by the author in tree different editions which are presented separately. The editions are compatible with various platforms and environments, to raise the degree of accessibility. The functionality supported by the tool is similar and available in all editions, however only presented once.

10.1 ECLIPSE PLUG-IN

This edition of the client is intended for the developers of software systems, of which requirements are managed by the tool. The requirements are then available in the developers IDE, which eliminates most probable technical and psychological obstacles.

10.1.1 Installation

The ReMaTo tool can be installed as any other Eclipse plug-in using the update manager[1] with '`http://remato.eide.biz/eclipse`' as the update site.

The selection of the plug-in features to be installed is illustrated in figure 10.1.

[1]Detailed Eclipse installation manual using the update manager is found at '`http://help.eclipse.org/help31/index.jsp?topic=/org.eclipse.platform.doc.user/tasks/tasks-34.htm`'

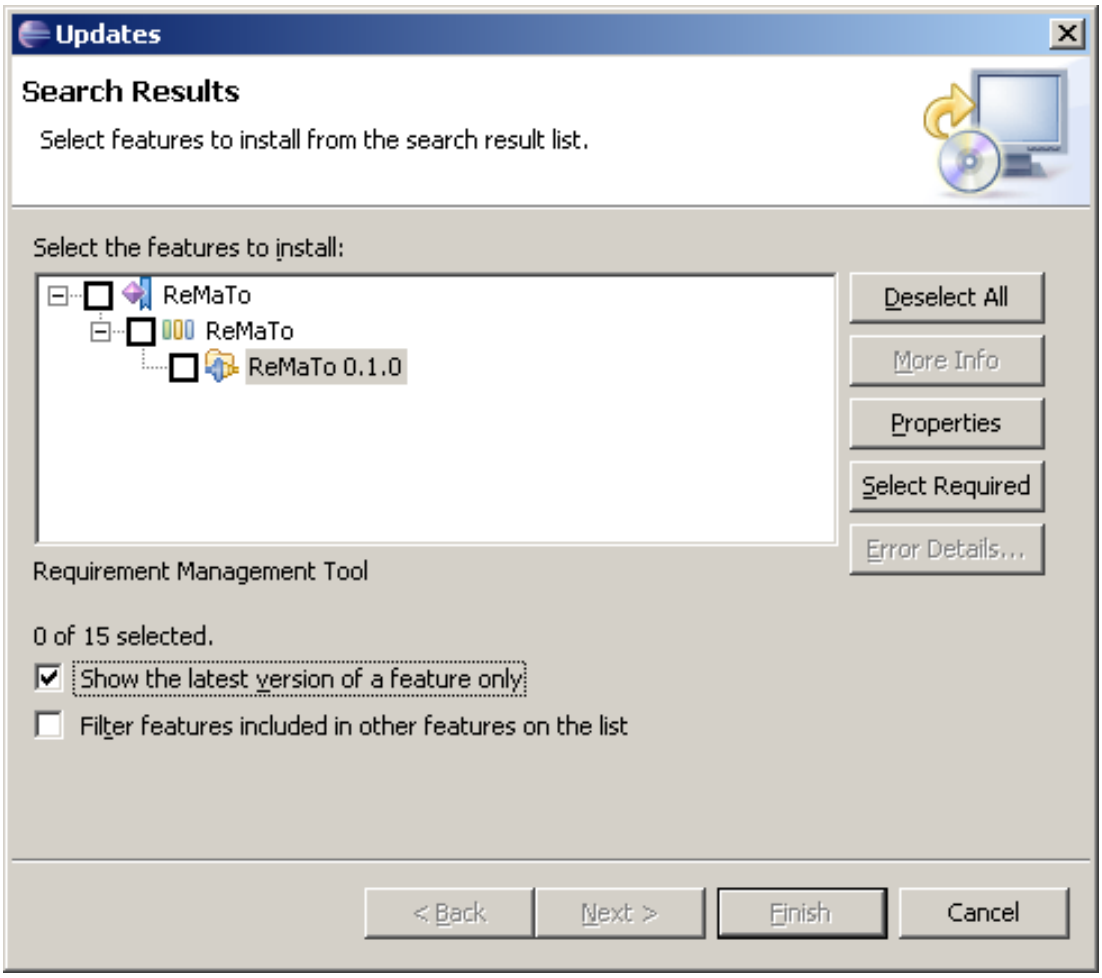

Figure 10.1: ReMaTo Eclipse plug-in installation

10.1.2 Login

To be able to restrict access, differentiate between different stakeholder roles and connect to multiple servers, a user session concept is introduced in the tool. The session is initiated by login, illustrated in figure 10.2, and terminated by logout.

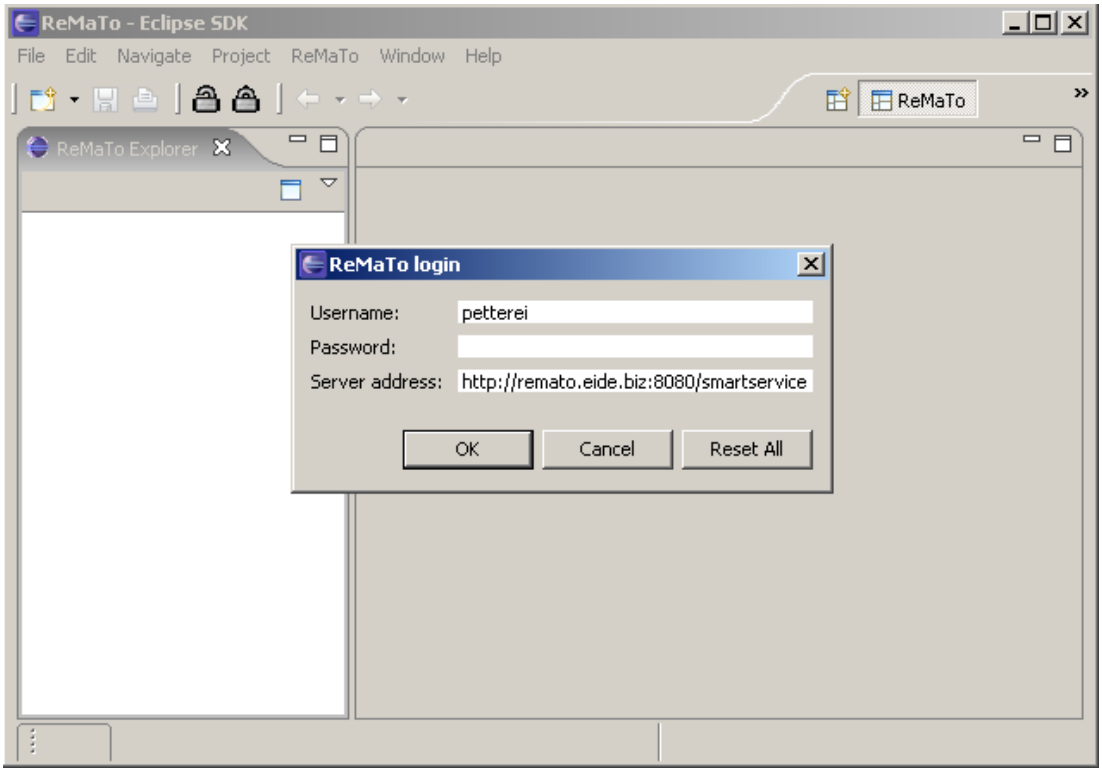

Figure 10.2: ReMaTo Eclipse plug-in login

10.1.3 ReMaTo Explorer

The tool's main view is the *ReMaTo Explorer*, which is illustrated to the left in figure 10.3. This view provides an outline of the domain concepts; projects, categories, requirement, stakeholders and requirement property templates (PropertySet).

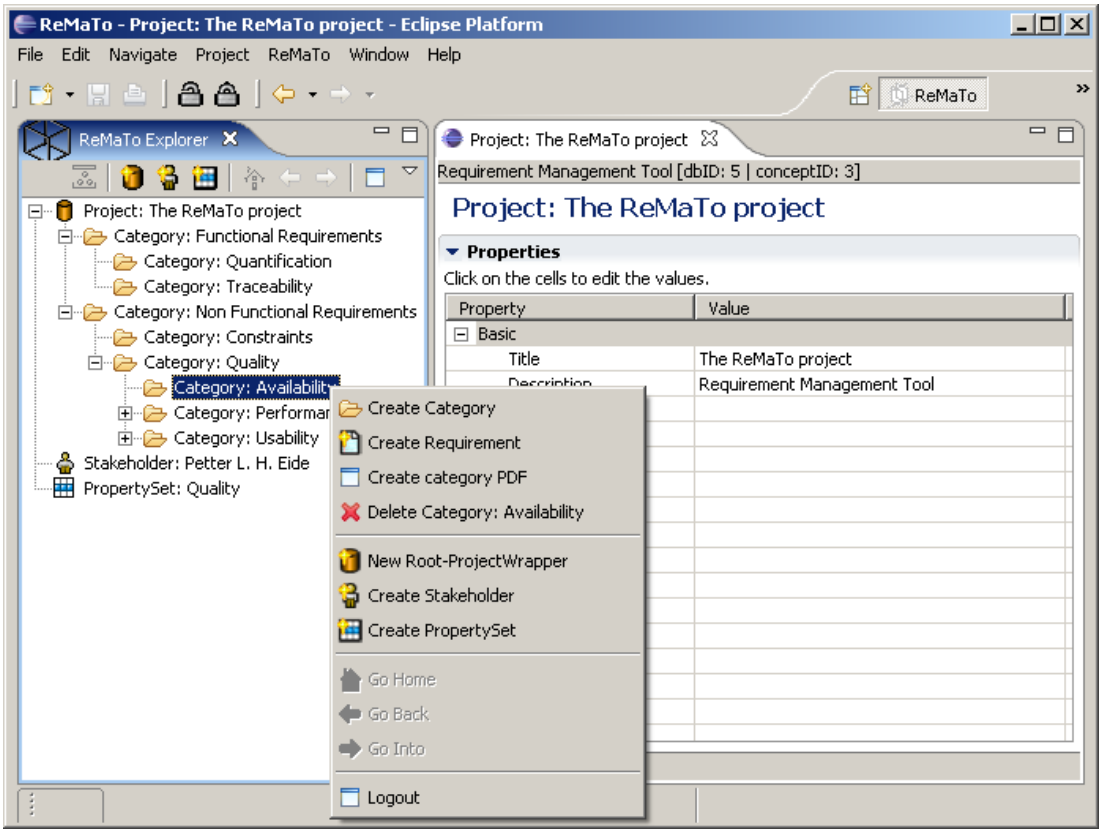

Figure 10.3: ReMaTo Explorer

Creation of domain concepts is triggered from the *ReMaTo Explorer* view by clicking buttons or selecting menu options. Creating of PDF reports is also available in this view. To view or edit a domain concept, a double click action on a concept will open the respective editor view. Such a view is illustrated to the right in figure 10.3.

10.1.4 Properties Template Editor

The *Properties Template Editor* provides functionality to define and update requirement property templates (PropertySets). The view is illustrated in figure 10.4.

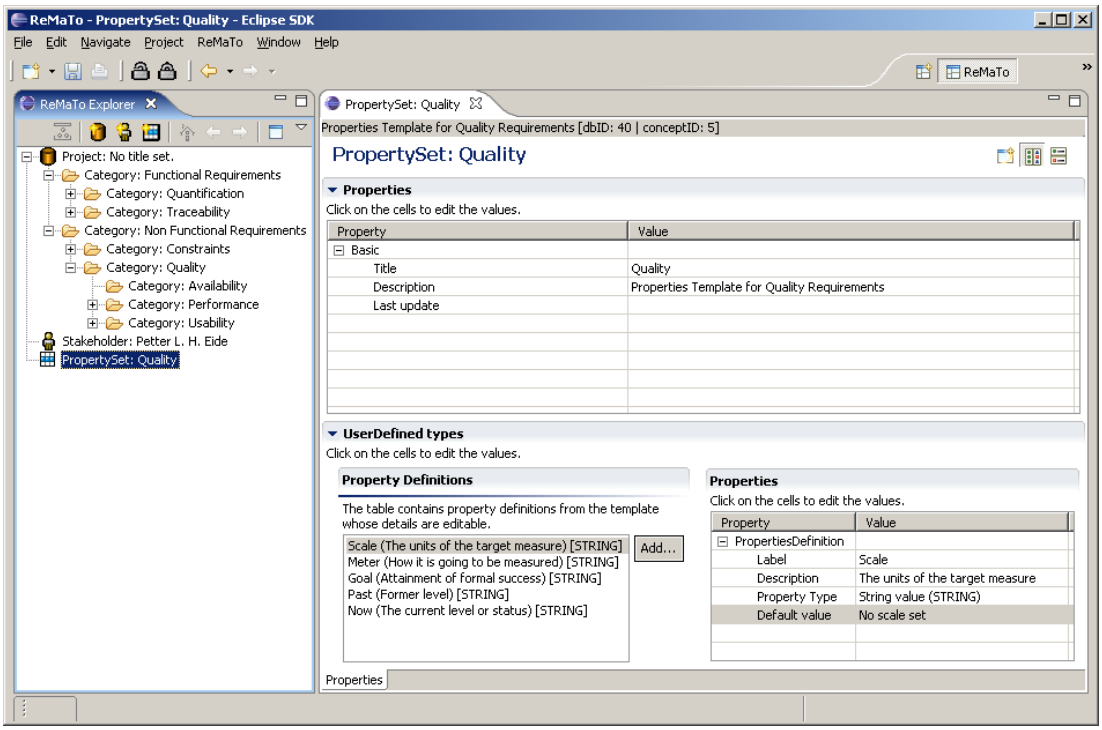

Figure 10.4: ReMaTo Properties Template Editor

As illustrated in the figure (10.4), defining and updating attached property definitions is supported. This adapts for utilizing various requirement taxonomies. The current properties in figure 10.4, reflects Gilbs quantification of requirements, as described in section 3.3. The property definitions' types can be string, integer, date etc. with appropriate validation of corresponding values. The underlying architecture is prepared for types as enumeration, stakeholder, requirements or other relations.

10.1.5 Category

The *Category* editor view is illustrated in figure 10.5. This view provides functionality to relate property templates to categories.

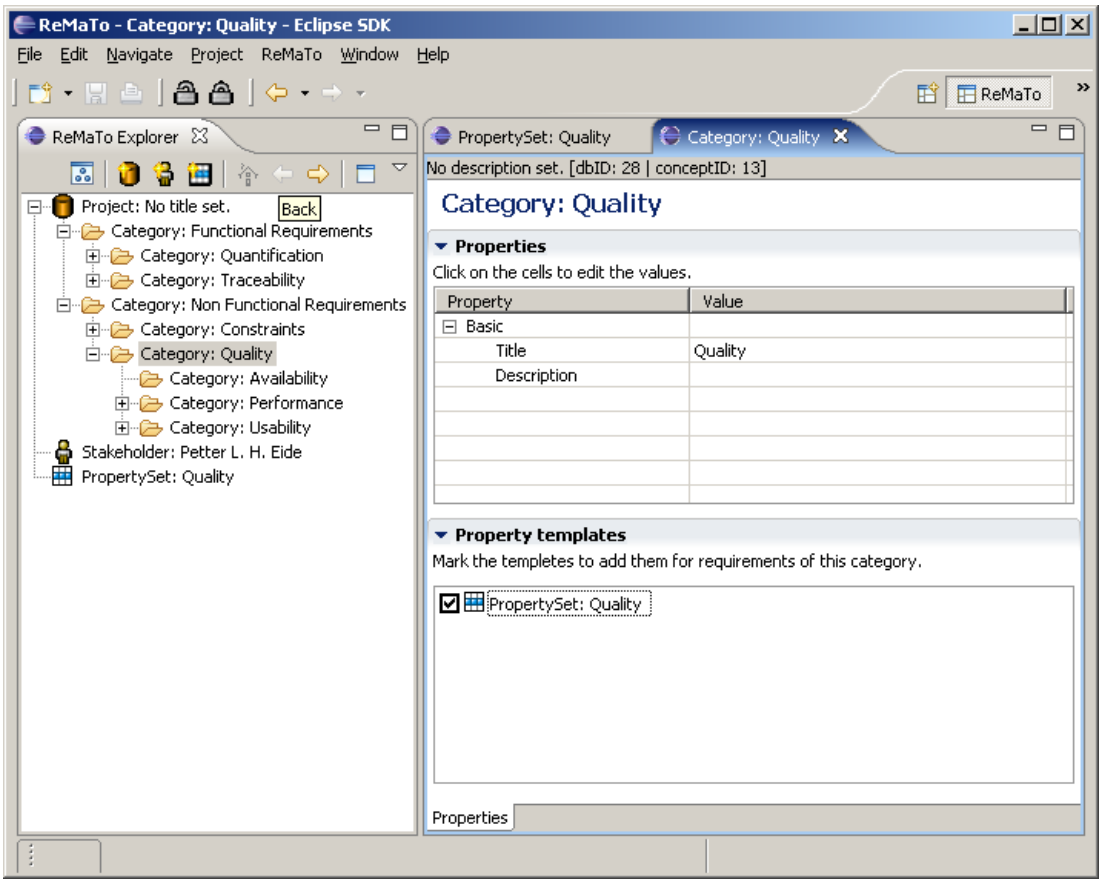

Figure 10.5: ReMaTo Category Editor

If a property template is related to a category, then requirements of this category will acquire the corresponding properties as defined in the property template.

10.1.6 Requirement

The requirement editor view is illustrated in figure 10.6. The different properties of the requirement are organized as a tree in the table.

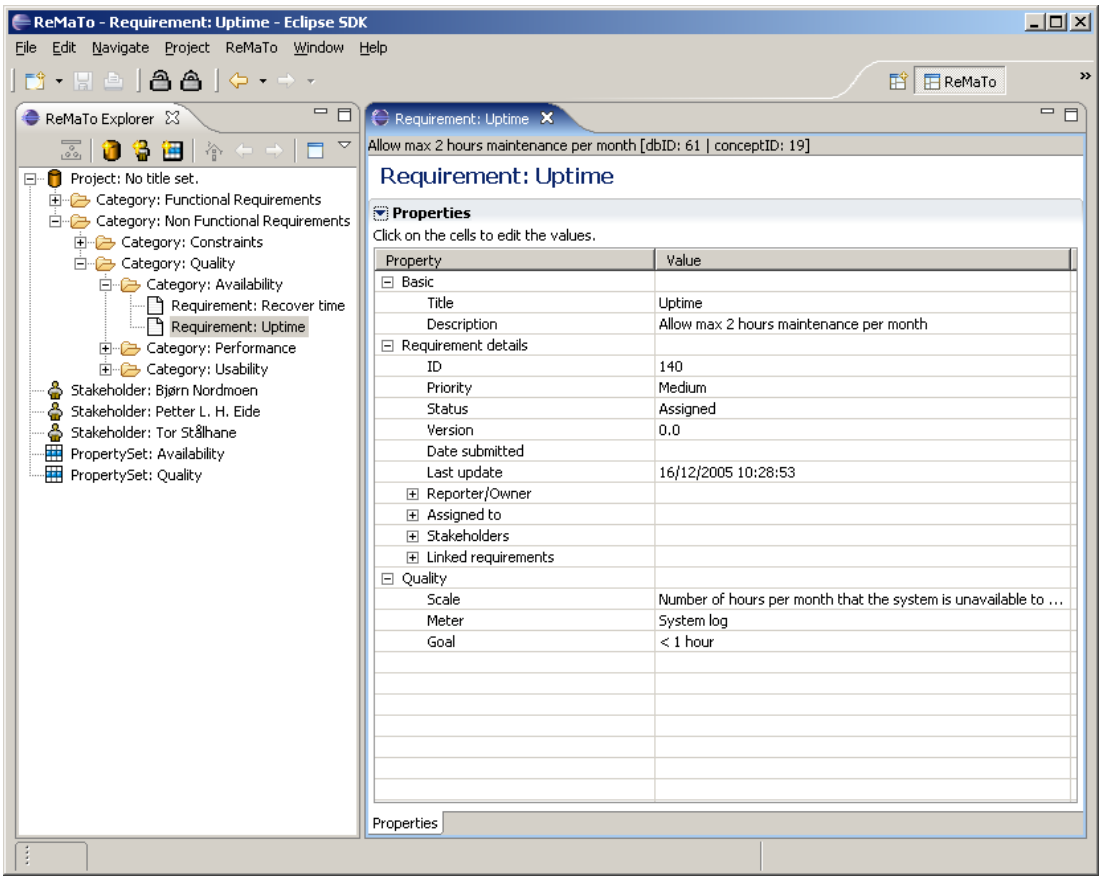

Figure 10.6: ReMaTo Requirements Editor

The predefined default properties of requirements, reflecting the domain model described in section 8.2, are found under the *Requirement details*-node. Relational attributes, such as *Stakeholders* or *Linked requirements*, can be expanded. Properties of property templates, which are attached to the parent category of the requirement, are displayed at the bottom of the table. The requirement in figure 8.2 belongs to the category "Availability", which has the property template "Quality" attached. Hence, the *Quality*-node with corresponding properties is displayed in the table editor.

In the current release of the plug-in, dependencies between requirements and to stakeholders can be reviewed. In future releases it will be possible to define and maintain

relations and dependencies to even more developer-related types of software artifacts.

10.1.7 Table View

The ReMaTo tool provides a table view to support displaying, in a compact and lucid manner, more than one requirement at the time. This view is illustrated in figure 10.7.

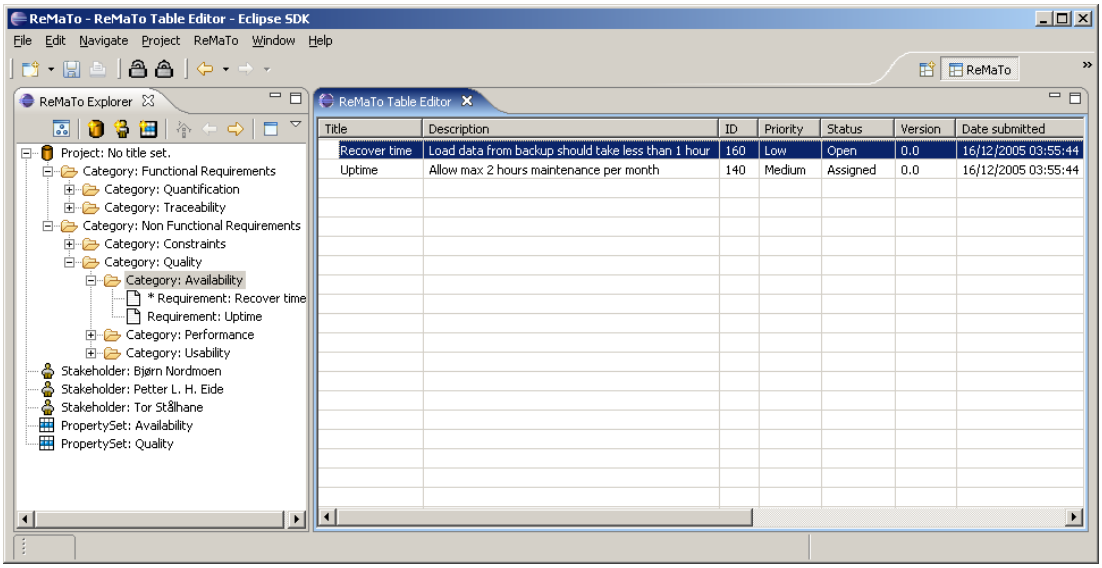

Figure 10.7: ReMaTo requirements table view

The view provides functionality to compare and sort all requirements of a category, and its subcategories.

10.1.8 Stakeholder

The stakeholder editor is illustrated in figure 10.8.

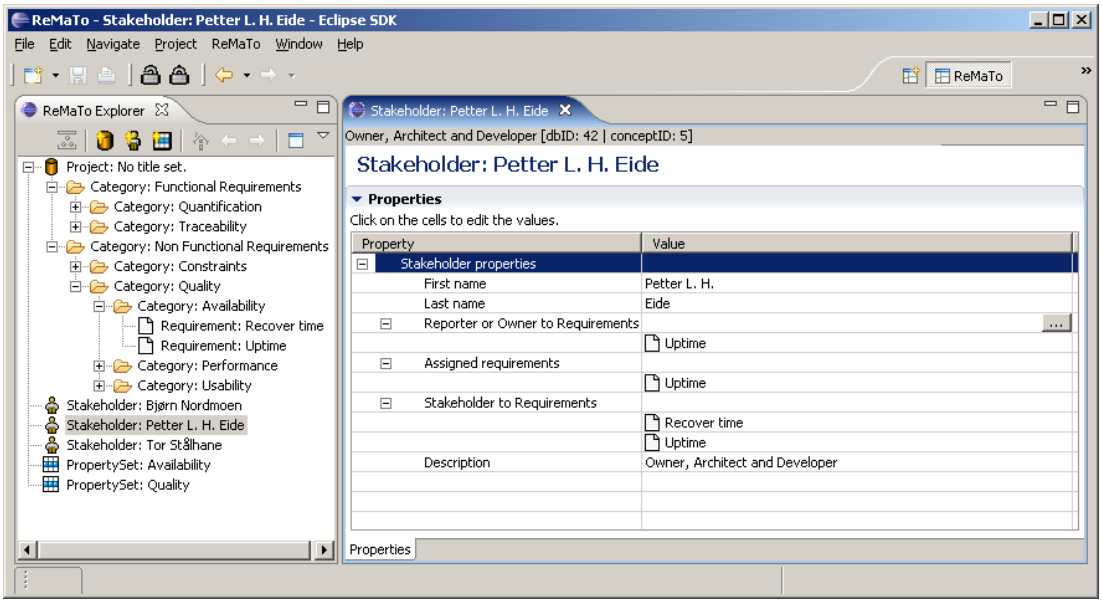

Figure 10.8: ReMaTo Stakeholder Editor

This view gives an overview of associated requirements and the corresponding specific role if any.

10.2 STANDALONE CLIENT

The standalone client edition provides the same functionality as the Eclipse plug-in edition does. The difference is that it can be installed without the Eclipse application, which current size is more than 100 MB. The standalone client edition's size is about 7 MB. The file structure of the standalone client is illustrated in figure 10.9.

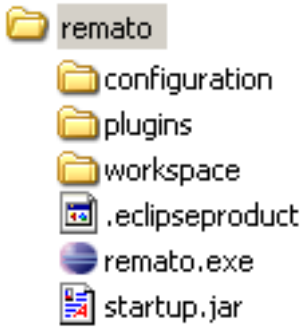

Figure 10.9: ReMaTo Standalone Client edition file structure

The target users of this edition are typically nontechnical stakeholders which do not have the Eclipse IDE installed. The standalone client edition is found on the enclosed CD or can be downloaded from 'https://remato.eide.biz'.

A sample screen shot from the standalone client edition is illustrated in figure 10.10.

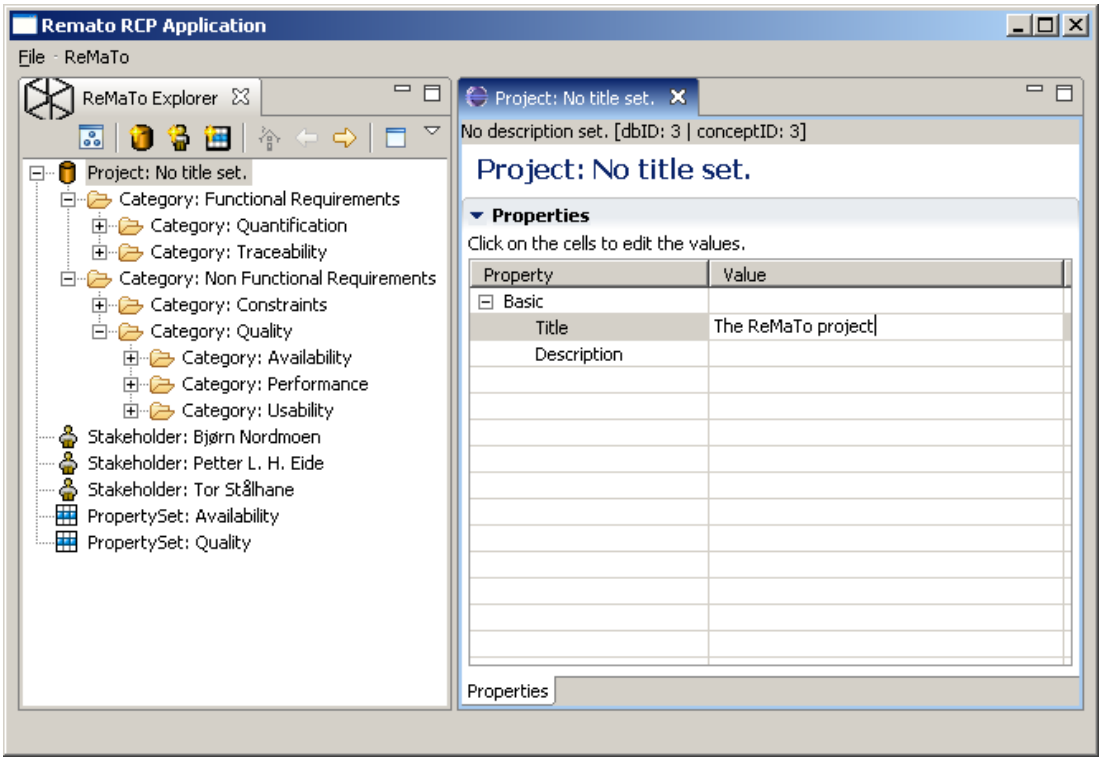

Figure 10.10: ReMaTo Standalone Client example

As the figure 10.10 illustrates, the Eclipse IDE specific menus and buttons are not available, while the ReMaTo functionality still is.

10.3 JAVA WEB START CLIENT

The "Java Web Start" (JWS) [18] standalone client edition of the ReMaTo tool is much like the ordinary standalone client described in previous section 10.2. The main difference is that is even more lightweight and is deployable with a single click over the network. Further it ensures the most current version of the application always will be deployed. Hence, if a binary is updated at the server, it will automatically be downloaded at next launch of the client. The JWS client edition is available from '`https://remato.eide.biz/jnlp/remato.jnlp`'.

The JWS deployment interface is illustrated in figure 10.11.

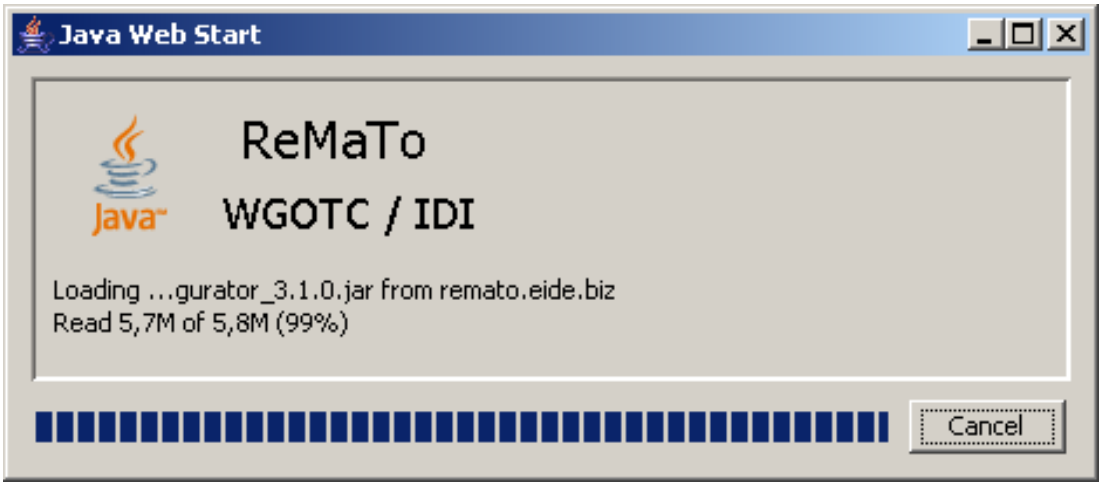

Figure 10.11: ReMaTo "Java Web Start" standalone client edition deployment

Part III

Evaluation and Discussion

Chapters

This part presents discussions about and evaluation of the project's contribution, the ReMaTo tool, and the project's progress and process. Issues and choices made in the project are covered. The purpose is to enlighten the results of the project.

CHAPTER 11

DISCUSSION

This chapter presents issues and decisions related to the ReMaTo software architecture.

11.1 REMATO PROTOTYPES EVOLUTION

The development of the tool, ReMaTo, has gone through several iterations and most of the source code has been rewritten / refactored at least twice. In total there have been developed three "prototype-series" of the tool, differentiated by major differences in the tool's underlying architecture.

The first prototype provided much of the same functionality as the latest version, except the framework providing PDF-exports. Another difference in functionality is that the property definitions were related directly to the categories. Hence, the concept *property templates* did not exist.

The first prototype's main architectural distinction, in comparison to later prototypes, is a more generic domain model and a "Domain Transfer Object" (DTO) pattern applied for exchanging information between server and client.

The second major prototype-series had a new and more concrete domain model. This model is shared between, and at the same time "extended" in various ways, at the server, as described in 9.3, and at the client, as described in section 9.2. The strategy for relating property definitions to the requirements is changed from directly attaching them to parent categories, to attach them to the new domain concept `PropertySet` (the requirement property/attribute template), which further is attached to the parent categories.

The last and current prototype or release introduces the optional proxy-layer at the client, providing seamless communication with the server.

11.2 DOMAIN MODEL

As described in previous section 11.1 the first prototype is based on a slightly different domain model than the later prototypes. This model is smaller and more generic, using the same pattern for defining attributes of the domain concepts as the client still does in the latest release; That is, use of property definitions and property values to define the attributes of requirements, categories and projects. Hence, no attributes are hard coded as fields, getters and setters.

These qualities make changes in the model less frequent or most unlikely to appear at all. In addition, the concepts can be handled more seamlessly all over the communication stack. Default templates of properties can be configured in files at server, or at client through a special user interface where special access would be needed. This yields the same behaviour as the today's more definite model, but with a major difference, core requirement properties could easily be overridden or customized by software developing organizations or teams.

This default requirements attribute template, provided by default configuration with a generic based domain model, or provided by reflection from a definite domain model, is an important quality of the tool, contributing it to become more easy-utilizing.

On the contrary to the more generic model, one can argue that a domain model should always be concrete, reflecting the concepts used in conversations and other work that a system is utilized in. This is also the main message of the book "Domain-driven design" [7] which this projects has used. The author agree to this, but personally still believe that different software developing companies and teams differ in what they consider as "concepts" of the requirements. If this tool is going to be flexible, then the author believe that it should not be leading in defining a requirements core properties or concepts, but allow the users to override this definition. Hence, a more generic model, still reflecting the main concepts of a requirement (however at another level of granularity), is appropriate.

All in all, the author personally votes for the more generic model, mainly due to the independence and flexibility it provides. The current release is based on the more definite model, due to constraints from the assigner of the project.

11.3 CLIENT-SERVER COMMUNICATION

As described in section 11.1 the first prototype based the exchanging of information between client and server on a DTO pattern. This pattern is realized by three different models of the domain; one for the server, one for exchanging (DTO) and one for the client. In addition there are four corresponding transformation algorithms. The transformation is performed once during each server-call, utilized only to exchange

information. The client and server make use of and maintain only their own representation of the model.

A positive consequence of this was the high degree of separation of concern. Once the DTO-model is designed, the client will be completely independent of the server and visa versa. This makes updates of the client applications in most cases optional and related to client issues only.

A drawback with the DTO-model is that if the model first is changed, this can result in corresponding need for changes in all three models and the coherent transformation algorithms, at worst in seven different source code areas. In return the model is smaller and more generic, as described in subsection 11.2, and the need for change will be less frequent or not needed at all.

The second major prototype introduces the definite domain model as described in section 11.1. This model is defined in the `remato.common` package, which both the client and server depends on. To the client, a wrapper-pattern is used and no transformation is needed. To the server, this is not possible, as explained in section 9.3, and a semi-automatic transformation is needed, which means that the transformation algorithm needs to be maintained if new domain concepts are introduced or dismissed.

At the client the wrapper-patterns results in some extra complexity, for instance the relations to other domain concepts. Memory leaks, in form of dual wrapper-instances for the same domain-model-object, are not wanted and will cause inconsistent behaviour of GUI-components. A parallel set of relations and hierarchies need therefore to be maintained with precision.

One can say that the DTO patterns are in a way still applied at the client-server-communication in the current release, but with a much stronger dependence to the common exchange model. This dependence does not give the advantages the DTO-pattern is supposed to. In contrast to the pure realization of the pattern in the first prototype, this realization is much more dependent on the common model, in that the client's and server's own model is directly dependent on this model. On the server, the transformation is as before. On the client, the transformation is now completely lazy, carried out just and only when a property value or relation is accessed.

In perspective and in contrast to the pure DTO-model, a change in today's model is more likely to occur. For instance if a new theory of defining properties of requirements is released, which probably will require different requirement properties in the domain model. If a change is needed in today's model, it will implicate comprehensive consequences. All deployed clients need to be updated in order to communicate with the server, and previously exported files will become outdated and most likely be imported incorrectly if needed.

11.4 SAVE HANDLING AND TRANSACTIONS

In the early prototypes, the principle “last save wins” is used. Thus, if a user retrieves a requirement from the server, modifies and saves it, other modifications on the same requirement in that time span could be overwritten.

In latest release, the principle still applies, but local transaction-handling is performed at the server in order to detect and avoid possible conflicts. Further, the session-proxy utilized at the client minimizes the age of information at the client, which minimizes the possibility of a conflict to occur.

The design of the session-proxy and the current client is also prepared for handling update-notifications from the server. Such notifications can, if implemented, avoid save-conflicts at the server completely by give users the opportunity to decide about conflicts in the same way as for instance modern CVS-repository-clients [4] does.

CHAPTER 12

EVALUATION

This chapter presents a brief evaluation of the ReMaTo tool, and some reflections and lessons learned during the projects course.

12.1 REMATO DEVELOPMENT

In accordance to the development methodology, described on page 7, testing has been performed continuously during the development of the tool. The tests of the server interface with and without the proxy-session have been performed automatically with TestNG [42] and JUnit [20]. The user interface and connected functionality have been tested by Bjørn Nordmoen and the author. The Eclipse Test and Performance Tools' Platform (TPTP) [44] has also been utilized by the author.

The stability of the tool has been fluctuating due to the extensive changes in architecture from iteration to iteration. The main functionality covered by the scope of this project is now in place (again) in the latest prototype released, with only minor adjustments outstanding.

In light of the purpose of this project, as described on page 3, the main goals concerning quantification and traceability are obtained. To implement functionality for advanced requirement traceability, such as history and baseline, is not in the primary scope of this project. The domain model is however prepared for such functionality to be added. The server can therefore with little effort be enabled to mark and save such information. How this information is to be retrieved and displayed at the client is not yet determined, however this is discussed more thoroughly in chapter 14.

12.2 PROJECT PROCESS AND PROGRESS

For future projects with similar context the author suggests two initiatives to improve the process.

The first initiative concerns efficiency. As mentioned, the changes from prototype to prototype are quite extensive in this project. The client and server have been rewritten at least twice. Some time and effort could have been saved or spent on extending the functionality provided, if these changes had been explored at another level. Extended use of architectural exploration and testing, like ATAM, CBAM [1] or others, could have given indications of the results before the complete prototypes were developed, and thus, possibly made the development of the discarded prototypes unnecessary.

The second initiative concerns interaction over geographical distances. It is important to identify and keep in mind the special challenges related to the development of a system at a remote site. For the developer, this means cultivate clear communication, and identify expectations and what is required, which the author believes is more challenging when mainly working geographically separated.

Part IV

Conclusion and Further Work

Chapters

This part presents the conclusion of the report and further work. The further work chapters encompass improvement and extension of the functionality of the RMT tool, ReMaTo, and topics of current and future interest concerning research in the traceability tool domain.

CHAPTER 13

CONCLUSION

Today, the use of requirement management tools, with support for quantification and traceability, is recognized as a significant capability in the software development and maintenance process, and as an important factor for the quality of the final software product.

The project has aimed to explore the technical aspects of such a tool. This has been done by implementing an adequate requirement management tool, ReMaTo. The tool it self, and this report concludes that it is feasible and doable to develop a lightweight, easy-utilizing RMT for general purpose use. The ReMaTo-tool can be held as a proof of this.

Making such a tool adaptive for most software developing companies and teams is however challenging. A way of meeting this is to provide customization functionalities which the ReMaTo tool does.

Making it available to all types of stakeholders in a software developing context is another challenge. This can be provided for by applying a client-server architecture, which this tool has.

The basic technical aspects are in other words clear. What remains is to look into ways of integrating such a tool into the business processes related to software development, and explore the implementation issues of advanced traceability to other software artifacts.

The integration with other business processes is not a part of this project. Implementation issues and further potential extensions are discussed in chapters 14 and 15.

CHAPTER 14

FURTHER WORK CONCERNING THE REMATO TOOL

For the specific tool, ReMaTo, developed in this project there exist several areas where the functionality can be extended.

The first area is enabling the history functionality of requirements, enabling business possibilities as described in section 4.1. A requirements history could for instance be available as a separate view, or as a spilt or tabbed view in the existing requirement editor view, which is illustrated in figure 10.6.

Next area target for extension is the implementation of functionality to support specification of relations from requirements to other software artifacts. The first step of this functionality is to enable it within the developers' IDE. This can be done by defining links as described in section 4.3.1. The challenging part here is to clarify how to identify unique, unambiguous path's to source code classes and packages. An easy, but not so flexible, solution would be to force project names to be equal to the root class-path, which is common for version-based repository projects. Another solution could be to create XML-based files in project folders which are enabled for tracing.

If the challenges connected to defining such relations are solved, a connected challenge is to maintain the relations. For IDE's with automatic refactoring functionalities, such as Eclipse, this can be done by adding rules to the refactoring algorithms that already exists. By doing so, a relation link between a requirement and a class, is automatically updated if the class is moved to another package. In multi-developer projects, this can be complex, and several issues must be clarified. Such as, when the link information should be updated in the requirements tool database, if even stored there; - on the instant moment the refactoring takes place, or when the developer commits the changes to the code-repository.

A third extension is to implement functionality to support specification of relations

to artifacts outside an IDE's environment. Such artifacts can be use cases, written in another tool, various documents etc. The challenge here lies in defining uniform access when the users need to access the links from different locations and different environments. A strategy to address this challenge could be to define a URL-syntax with support for validation.

At present time, the only IDE supported by ReMaTo is Eclipse. If the tool shall reach the general purpose market for requirement management, plug-ins for other IDEs, such as NetBeans [30], needs to be written. The componentization of the client source code will ease this work.

CHAPTER 15

FURTHER WORK RELATED TO THE DOMAIN OF TRACEABILITY

Several areas, concerning topics related to traceability between requirements and other software artifacts, remains to be fully explored. Related to tool support, the following paragraphs present the most relevant issues.

Storage

Traceability information needs a place to be stored, either in the RMT, an external database, a file, a model, or other. The key challenges lies in defining the traces and providing automatic maintenances. The storage of the traces is central in finding a solution to these challenges. The existing theories differentiate between intra-model and extra-model traceability storage.

The intra-model approach populates the artifacts itself with traceability information, e.g. by using tags, properties. In UML, it could be done with stereotypes and tagged values. In Java it can be done with tags in java-doc areas. Such population of traceability information in the artifact itself could lead to a pollution of the artifact, and is further not applicable to all types of relations such as relations to external document with unknown or unmodifiable format.

The extra-model approach leaves the original artifact or model untouched and store the trace information in an external model. The theory looks into possible mechanisms to reference into the target artifact or model by using unique identifiers which is not affected by changes concerning the targeted artifact. If such references and models is possible, the issue may be reduced to a tool implementation issue only. Then each tool vendor may implement this differently, whilst ideally supporting a "standard traceability scheme". Else a shared additional artifact or model is needed to maintain

the references, which further can be used as a proxy for traces into the real artifact or model.

Ownership

The ownership of traceability links or other traceability information is an issue related to the traceability storage challenges. If several tools are using the traces, it may become uncertain who the owner is. Theories describe this auditability of traceability links as an aspect of tracking meta-information of traceability. Further, it can be used to assessing information like: when was a traceability link induced, how (by transformation, manually), why etc.

Metrics

If using traceability by tool support to analyze or verify software, then the tools it selves need to be validated as well. The tools are dependent of the traceability information. Hence, metrics-definition and identification for traceability should be explored.

Business Applications

As the described in the motivation section, on page 2, and further in section 4.4, traceability can give an intangible added value to software developing companies. For instance resources used on maintenance, which can be reduced by traceability provided by tools, e.g. through impact analysis, improving the time spent on locating change locations, bugs and errors etc. Traceability may be used for consistency checking in such work. Even if the technology is available, it is still a big challenge to adopt it and integrate it to existing business processes.

Terminology Issues

To encourage interoperability among the variety of tools developed and the research in progress, a need for establishing terminology in the traceability area is arising, i.e. to have more precise definitions of the terms.

APPENDIX A

PROJECT PLAN

A.1 MILESTONES

A.1.1 The Requirement Tool

M1: 05.10.09 v0.1

1. Eclipse based client with new domain model
2. Flexible Project and Category hierarchy with Drag-n-Drop functionality
3. Eclipse plug-in update-site

M2: 05.10.16 v0.2

1. Requirements with flexible user-defined properties templates
2. Flexible means date, string, integer
3. Domain-model functionality for advanced versioning
4. Domain-model support baseline versions.

M3: 05.10.23 v0.3

1. XML based import/export
2. PDF report generation
3. CRUD functionality for stakeholders.

M4: 05.10.30 v1.0

1. Standalone RCP-client
2. Web-based client

M5: 05.11.06 v1.1

1. Simple traceability functionality.
2. Visualization of requirements dependencies/links?

M6: 05.11.13 v2.0

1. Advanced traceability functionality.

M7: 05.12.20 Deploy latest releases

1. Eclipse plug-in client
2. Standalone RCP client
3. Web-based client

M6 will be the last milestone to add new functionality.

A.1.2 The Report

M1: 05.10.09 Introduction

- Detailed outline for this chapter and most of the content
- Rough outline for prestudy chapter

M2: 05.10.16 Prestudy

- Get necessary papers and readings

M3: 05.10.23 Prestudy

- Halfway prestudy

M4: 05.10.30 Prestudy

- Finished prestudy
- Rough outline for contribution and rest of the report

M5: 05.12.20 Report delivery

APPENDIX B

RMT SRS

The complete SRS of the Requirement Management tool, written by Bjørn Nordmoen, is included in the following pages.

Appendix: Requirements for a Requirements Tool

This appendix lists the requirements defined for the requirements tool. The requirements have been handled using the modified version of Mantis.

1. Business Requirements

1.1. Cost

1.1.1. Capital Investment

21: Cost of a new version					
Priority	normal	**Status**	new	**Version**	0.1
Category	B_Cost_Capital Investment		**Type**	Business Requirement	
Date Submitted	01-24-05 15:20		**Last Update**	09-07-05 10:39	
Reporter	Bnordmoen		**Assigned To**		
Stakeholders	ALL				
Description	The cost of a new version Server installation: < 100 USD Each client license: < 19 USD				
Workpackages					
Links					

1.1.2. Support and Maintenance Cost

22: Upgrade cost					
Priority	normal	**Status**	new	**Version**	0.1
Category	B_Cost_Support and Maintenance Cost		**Type**	Business Requirement	
Date Submitted	01-24-05 15:21		**Last Update**	09-07-05 10:40	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	ALL				
Description	The cost to upgrade to a newer version < 19USD				
Workpackages					
Links					

2. Functional Requirements

2.1. CRUD

5: Create, Update and Delete Requirements					
Priority	high	**Status**	new	**Version**	0.1
Category	F_CRUD		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:35		**Last Update**	01-18-05 11:45	
Reporter	tneple		**Assigned To**		
Stakeholders	Reporter				
Description	Users having the Reporter role should be able to create a new requirement. They should also be able to update and delete a requirement.				
Links	related to	0000028	new	Create requirement	

	related to	0000025	new	Login and view summary

8: Links to other requirements

Priority	high	**Status**	new	**Version**	0.1
Category	F_CRUD		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:45		**Last Update**	01-18-05 11:45	
Reporter	tneple		**Assigned To**		
Stakeholders					
Description	It should be possible for a Reporter to create links from a requirement to one or more requirements.				
Links					

7: Sorting

Priority	high	**Status**	new	**Version**	0.1
Category	F_CRUD		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:43		**Last Update**	01-18-05 11:43	
Reporter	tneple		**Assigned To**		
Stakeholders					
Description	It should be possible to produce a sorted view of the requirements. It should be possible to sort by any one (or a combination) of the defined fields.				
Links	related to	0000026	new	View requirements list sorted	

6: Standard requirement fields

Priority	high	**Status**	new	**Version**	0.1
Category	F_CRUD		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:41		**Last Update**	01-18-05 11:41	
Reporter	tneple		**Assigned To**		
Stakeholders	All				
Description	The following information has to be stored for each requirement - Unique ID - Date created - Created by - Responsible - Priority - Category (Hierarchical) - Status - Scale - Meter (optional) - Goal (date and scale qualifiers) - Past (date and scale qualifiers) - MODELWARE specific fields (can be created by custom fields functionality) - MODELWARE partner (can use created by?) - Needed for scenario - Link to work packages - Technical area				
Links					

2.2. Customisation

15: Custom fields					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Customisation		**Type**	Functional Requirement	
Date Submitted	01-18-05 12:05		**Last Update**	01-18-05 12:05	
Reporter	tneple		**Assigned To**		
Stakeholders	Administrator				
Description	It should be possible for the administrator to add custom fields to the definition of requirements for a project. Each custom field should have a name and a type. The following types of fields should be supported: - Text - Enum - Boolean				
Links	related to 0000030 new Easy to add use defined properties to the requirements template				

2.3. Data Exchange

16: Export/import requirements data from external formats					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Data exchange		**Type**	Functional Requirement	
Date Submitted	01-18-05 14:21		**Last Update**	01-18-05 14:21	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	Reporter, administrator				
Description	Should support the following formats: - CSV - XML (needs to be defined) Should also import directly from email attachments				
Links					

2.4. Manage Users and Roles

13: Create, update, and delete users					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Manage users and roles		**Type**	Functional Requirement	
Date Submitted	01-18-05 12:02		**Last Update**	01-18-05 12:18	
Reporter	tneple		**Assigned To**		
Stakeholders	Administrator				
Description	It should be possible for the administrator to manage users in the system, This includes creating, updating, and deleting user accounts and assigning the users to defined roles.				
Links	related to 0000012 new Define roles and rights				

14: Create, update, and delete projects					
Priority	normal	**Status**	new	**Version**	0.1
Category	F_Manage users and roles		**Type**	Functional Requirement	
Date Submitted	01-18-05 12:03		**Last Update**	01-18-05 12:03	
Reporter	tneple		**Assigned To**		
Stakeholders	Administrator				

Description	It should be possible for the administrator to manage a set of projects within the requirements tool.
Links	

12: Define roles and rights					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Manage users and roles		**Type**	Functional Requirement	
Date Submitted	01-18-05 12:00		**Last Update**	01-18-05 12:00	
Reporter	tneple		**Assigned To**		
Stakeholders	Administrator				
Description	It should be possible to define a set of roles with assigned rights Rights - Create - Update - Delete - Read only - Administrator (all rights)				
Links	related to 0000013 new Â Create, update, and delete users				

2.5. Report

11: Create report based on version					
Priority	normal	**Status**	new	**Version**	0.1
Category	F_Report		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:56		**Last Update**	01-18-05 11:56	
Reporter	tneple		**Assigned To**		
Stakeholders	Reporter, Viewer, Administrator				
Description	It should be possible for a user to create a report from the requirements based on: - a date snapshot (the set of requirements at a specific date) - a release name				
Links					

10: Report output format					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Report		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:53		**Last Update**	01-18-05 11:53	
Reporter	tneple		**Assigned To**		
Stakeholders	Reporter, Viewer, Administrator				
Description	It should as a minimum be possible to create reports in HTML format.				
Links					

2.6. Version Handling

4: Release version					
Priority	normal	**Status**	new	**Version**	0.1
Category	F_Version Handling		**Type**	Functional Requirement	

Date Submitted	01-18-05 11:18	**Last Update**	01-18-05 11:20
Reporter	tneple	**Assigned To**	
Stakeholders	Reporter, Viewer, Administrator		
Description	It should be possible to tag a set or snapshot of requirements as a release version. A release should be given a name.		
Links			

3: Changelog					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Version Handling		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:16		**Last Update**	01-18-05 11:16	
Reporter	tneple		**Assigned To**		
Stakeholders	Reporter, Viewer				
Description	Each requirement should be "date stamped" when created and updated. This information should be viewable as a requirement change log.				
Links					

2: Unique ID					
Priority	high	**Status**	new	**Version**	0.1
Category	F_Version Handling		**Type**	Functional Requirement	
Date Submitted	01-18-05 11:12		**Last Update**	01-18-05 11:13	
Reporter	tneple		**Assigned To**		
Stakeholders	Reporter				
Description	Each requirement should have a unique identifier that never changes. The identifier should be assigned to the requirement automatically				
Links					

3. Quality Requirements

3.1. Adaptability and Flexibility

30: Easy to add use defined properties to the requirements template					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Adaptability_Flexibility		**Type**	Quality Requirement	
Date Submitted	02-11-05 10:33		**Last Update**	09-07-05 10:22	
Reporter	MODELWARE		**Assigned To**		
Stakeholders					
Description	Easy to add use defined properties to the requirements template				
Scale	Time in sec. to add a user-defined property to a requirement				
Meter	Stopwatch				
Goal	30 sec				
Past	?				
Links	related to	0000015	new	Custom fields	

3.2. Availability

19:Â Uptime					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Availability		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:13		**Last Update**	09-07-05 10:23	
Reporter	bnordmoen		**Assigned To**		
Stakeholders					
Description	Allow for two hours backup or maintenance a week				
Scale	Number of hours per week the system is available for requirements work				
Meter	Stopwatch				
Goal	> 166 hours				
Past	?				
Links					

3.2.1. Recoverability

20: Recover for database crash					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Availability_Recoverability		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:17		**Last Update**	09-07-05 10:25	
Reporter	bnordmoen		**Assigned To**		
Stakeholders					
Description	Ideally, there should be an email sent to the administrator when the database crashes.				
Scale	Time in hours from problem discovered to system up running from a backup				
Meter	system log (stopwatch)				
Goal	< 1 hour				
Past					
Links					

3.3. Capacity

29: Number of users, projects and requirements					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Capacity		**Type**	Quality Requirement	
Date Submitted	01-24-05 16:23		**Last Update**	09-05-05 13:44	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	ALL				
Description	The minimum capacity of the system when it comes to number of users, projects, and requirements per project				
Scale	All availability and usability req. still to be valid with 100 users, 100 project with 100 req. each				
Meter					
Goal	minimum 100 users, 100 projects with 100 requirements each				
Past					
Links					

3.4. Usability

3.4.1. Learnability

24: Time to learn					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Usability_Learnability		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:26		**Last Update**	09-07-05 10:28	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	Reporter, Viewer, Administrator				
Description	Time to learn the features of the tool.				
Scale	Time in hours self study on a test project using the tool itself				
Meter	Stopwatch				
Goal	< 1/2 hour for a viewer, < 1 hour for a reporter, < 2 hours for an administrator				
Past					
Links					

3.4.2. User Productivity

25: Login and view summary					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Usability_User productivity		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:37		**Last Update**	09-07-05 10:46	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	ALL				
Description	Login in to system and get the initial requirements summary page displayed				
Scale	Time in sec from executing the login form to initial summary page is displayed				
Meter	Stopwatch				
Goal	< 20 sec.				
Past					
Links	related to 0000005 new Â Create, Update and Delete Requirements				

26: View requirementslist sorted					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Usability_User productivity		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:39		**Last Update**	09-07-05 10:34	
Reporter	bnordmoen		**Assigned To**		
Stakeholders					
Description	View requirements list sorted on the main fields				
Scale	Time in sec after logged in to sorted list is displayed				
Meter	Stopwatch				
Goal	< 20 sec				
Past					
Links	related to 0000007 new Sorting				

28: Create requirement					
Priority	normal	**Status**	new	**Version**	0.1
Category	Q_Usability_User productivity		**Type**	Quality Requirement	
Date Submitted	01-24-05 15:42		**Last Update**	09-07-05 10:33	
Reporter	bnordmoen		**Assigned To**		
Stakeholders	Reporter				
Description	Time to open the requirements form for a new requirement or an update				
Scale	Time in sec. to open the requirements form after logged in.				
Meter	Stopwatch				
Goal	< 10 sec				
Past					
Links	related to 0000005 new Create, Update and Delete Requirements				

APPENDIX C

CLASS DIAGRAMS

Additional class diagrams.

C.1 THE `remato.common.session` PACKAGE

The figure C.1 illustrates the `remato.common.session` package.

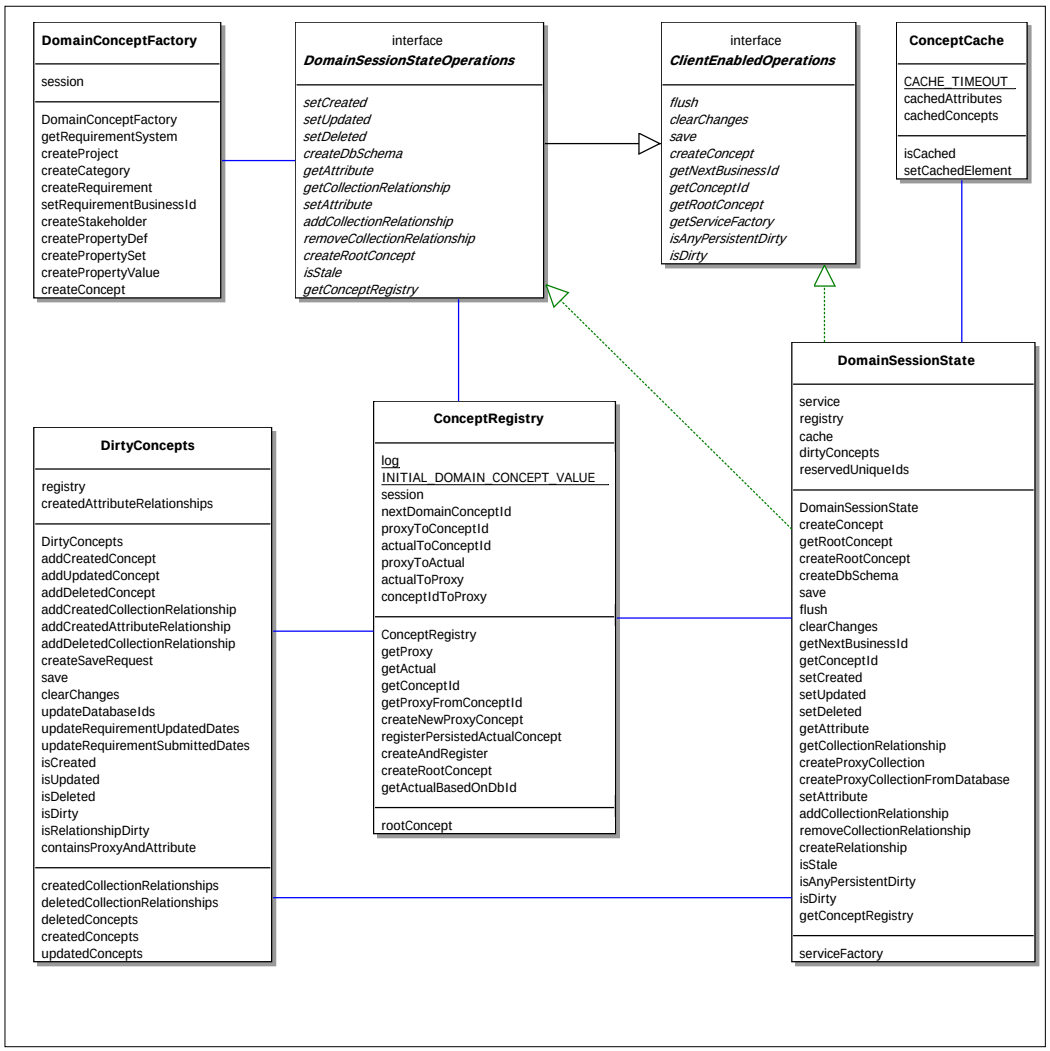

Figure C.1: The `remato.common.session` package

BIBLIOGRAPHY

[1] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1998.

[2] *Borland*. `http://www.borland.com/`. Software Optimization Tools Provider.

[3] *cglib*. `http://cglib.sourceforge.net/`. Code Generation Library.

[4] *CVS*. `http://www.nongnu.org/cvs/`. Concurrent Versions System.

[5] *Eclipse*. `http://www.eclipse.org/`. Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software.

[6] *European Conference on Model Driven Architecture - Foundations and Applications*. `http://www.ecmda-fa.org/2005/`. November 7-10th 2005, Nuremberg, Germany.

[7] Eric Evans. *Domain-Driven Design*. Addison-Wesley, 2004.

[8] Donald G. Firesmith. Common concepts underlying safety, security, and survivability engineering. Technical Note CMU/SEI-2003-TN-033, Carnegie Mellon Software Engineering Institute, 2003.

[9] *Sintef sentral i europeisk forskningsprosjekt*. `http://www.cw.no/index.cfm/ill_liste/artikkel/id/51643`. Seismikkselskapet WesternGeco Schlumberger bidrar til å finne metoder som hindrer at programkoden i større it-løsninger forfaller.

[10] Martin Fowler and Kendall Scott. *UML distilled (2nd ed.): a brief guide to the standard object modeling language*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, July 2001.

[11] Tom Gilb. *Principles of Software Engineering Management*. Addison-Wesley, 1988.

[12] Tom Gilb. *Competitive Engineering*. Elsevier Butterworth-Heinemann, 2005.

[13] O. C. Z. Gotel and A. C. W. Finkelstein. An analysis of the requirements traceability problem. In *First International Conference on Requirements Engineering (ICRE)*, pages 94–101. IEEE Computer Society Press, april 1994.

[14] *Hessian*. `http://www.caucho.com/hessian/`. Binary Web Service Protocol.

[15] Ieee standard glossary of software engineering terminology, 1990. IEEE Std 610.12-1990.

[16] Ieee recommended practice for software requirements specification, ieee std, 1998. Revision of IEEE Std 830-1998.

[17] 2000. ISO 9126: Software Software engineering.

[18] *Java Web Start Technology*. `http://java.sun.com/products/javawebstart/`. Support standalone Java software applications to be deployed with a single click over the network.

[19] C.G. Mikael Johansson. *Social and organizational aspects of requirements engineering methods*. PhD thesis, 1999. Avhandling (fil. dr.) - Linköpings universitet, 1999.

[20] *JUnit*. `http://www.junit.org/`. A regression testing framework.

[21] Philippe Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[22] Søren Lauesen. *Software Requirements: Styles and Techniques*. Addison-Wesley, 2002.

[23] Dean Leffingwell and Don Widrig. *Managing Software Requirements*. Addison-Westley, 2000.

[24] P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. McGraw-Hill, 1995.

[25] Richard J. Malak and Christiaan J. J. Paredis. Foundations of validating reusable behavioral models in engineering design problems. In *Winter Simulation Conference*, pages 420–428, 2004.

[26] Raimundas Matulevicius. *Process Support for Requirements Engineering A Requirements Engineering Tool Evaluation Approach*. PhD thesis, NTNU, 2005. Doctoral theses at NTNU, 2005:142.

[27] J. A. McCall and M.T. M. T. Matsumoto. Quality factors. In *Software Quality Measurement Manual, Vol. II.* Rome Air Development Center, 1980.

[28] *Model Driven Development integration (MDDi)*. `http://www.eclipse.org/mddi/`. The MDDi project produces an extensible framework and exemplary tools dedicated to integration of modeling tools in Eclipse.

[29] *ModelWare*. `http://www.modelware-ist.org/`. Large-scale deployment of Model Driven Development.

[30] *NetBeans*. `http://www.netbeans.org/`.

[31] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 35–46, New York, NY, USA, 2000. ACM Press.

[32] *Oxford Dictionary*. The Concise Dictionary Eleventh Edition.

[33] Frauke Paetsch, Armin Eberlein, and Frank Maurer. Requirements engineering and agile software development. In *WETICE '03: Proceedings of the Twelfth International Workshop on Enabling Technologies*, page 308, Washington, DC, USA, 2003. IEEE Computer Society.

[34] Francisco A. C. Pinheiro. Formal and informal aspects of requirements tracing.

[35] Francisco A. C. Pinheiro and Joseph A. Goguen. An object-oriented tool for tracing requirements. *IEEE Softw.*, 13(2):52–64, 1996.

[36] Klaus Pohl. Requirements Engineering: An Overview. Technical Report AIB-05-1996, RWTH Aachen, 1996.

[37] Norah Power and Tony Moynihan. A theory of requirements documentation situated in practice. In *SIGDOC '03: Proceedings of the 21st annual international conference on Documentation*, pages 86–92, New York, NY, USA, 2003. ACM Press.

[38] Suzanne Robertson and James Robertson. *Mastering the requirements process*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999.

[39] I. Sommerville and G. Kotonya. *Requirements Engineering. Processes and Techniques.* John Wiley and Sons., 1998.

[40] I. Sommerville and P. Sawyer. *Requirements Engineering. A Good Practice.* John Wiley and Sons., 1997.

[41] *SourceForge.net*. `http://SourceForge.net`. The world's largest Open Source software development web site.

[42] *TestNG*. `http://testng.org/doc/`. A testing framework.

[43] *TeXnicCenter*. `http://www.toolscenter.org/`. IDE for developing LaTeX-documents.

[44] *TPTP*. `http://www.eclipse.org/tptp/`. Test and Performance Tools Platform.

[45] *WesternGeco*. `http://www.westerngeco.com/`. A Schlumberger company.

[46] *Wikipedia*. `www.wikipedia.org`. The Free Encyclopedia.

[47] Pamela Zave. Classification of research efforts in requirements engineering. *ACM Comput. Surv.*, 29(4):315–321, 1997.

[48] Andrea Zisman, George Spanoudakis, Elena Pérez-Miñana, and Paul Krause. Tracing software requirements artifacts. In *Software Engineering Research and Practice*, pages 448–455, 2003.

GLOSSARY

Notation	Description
ATAM	Architecture Tradeoff Analysis Method, 82
BTA	Borland Together Architect 2006, 10
CA	Certificate Authority, 10
CBAM	Cost Benefit Analysis Method, 82
COTS	Commercial Off-The-Shelf, 3
CRUD	Create, Retrieve, Update and Delete functionality, 34
elicit	in requirement context; find or draw out the requirements from stakeholders and target environment, 1
IDE	Integrated Developer Environment, 9
jar	Java ARchives, 10
LOC	Lines Of Code is the number of lines of code in a namespace, classifier or method, including comments and white-lines., 57
NF	Non Functional (Requirement), 20
NOC	Number Of Classes counts the number of classes. , 57
NOIS	Number Of Import Statements counts the number of imported packages /classes. , 57
NOM	Number Of Members counts the number of members, i.e. attributes and operations. Inherited members can optionally be included in the total., 57

Notation	Description
NOO	Number Of Operations counts the number of operations. Inherited operations may be counted optionally., 57
PIS	Package Interface Size is the number of classes in a package that are used from outside the package. A class uses a namespace if it calls methods, accesses attributes or extends a class declared in that namespace. , 57
PS	Package Size is the number of classes which are defined in the measured package . Inner classes are not counted. , 57
quantification	identification of measurable, mutually properties of similar requirements and the elicitation of corresponding values for each requirement, 24
RE	requirements engineering, 1
refactor	in software development, the term is used for changing or rewriting existing source code or design, 87
requirement attributes	meta-data (definition) and data (value) that describes a requirement, 3
requirements engineering	an activity which aims at discovering, documenting and maintaining a set of requirements, 13
RMT	requirement management tool, 3
SE	software engineering, 1
SRS	software requirements specification, 20
traceability	the ability to relate artifacts which are created during the development of a software system (e.g., requirements, design and code artifacts) with each other, the stakeholders that created them, and/or the rationale underpinning their exact form, 29

INDEX